CAN and LIN

Automotive Diagnostic Command Set User Manual

July 2014
372139G-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *NI Services* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code feedback.

© 2007–2014 National Instruments. All rights reserved.

Legal Information

Limited Warranty

This document is provided ‘as is’ and is subject to being changed, without notice, in future editions. For the latest version, refer to `ni.com/manuals`. NI reviews this document carefully for technical accuracy; however, NI MAKES NO EXPRESS OR IMPLIED WARRANTIES AS TO THE ACCURACY OF THE INFORMATION CONTAINED HEREIN AND SHALL NOT BE LIABLE FOR ANY ERRORS.

NI warrants that its hardware products will be free of defects in materials and workmanship that cause the product to fail to substantially conform to the applicable NI published specifications for one (1) year from the date of invoice.

For a period of ninety (90) days from the date of invoice, NI warrants that (i) its software products will perform substantially in accordance with the applicable documentation provided with the software and (ii) the software media will be free from defects in materials and workmanship.

If NI receives notice of a defect or non-conformance during the applicable warranty period, NI will, in its discretion: (i) repair or replace the affected product, or (ii) refund the fees paid for the affected product. Repaired or replaced Hardware will be warranted for the remainder of the original warranty period or ninety (90) days, whichever is longer. If NI elects to repair or replace the product, NI may use new or refurbished parts or products that are equivalent to new in performance and reliability and are at least functionally equivalent to the original part or product.

You must obtain an RMA number from NI before returning any product to NI. NI reserves the right to charge a fee for examining and testing Hardware not covered by the Limited Warranty.

This Limited Warranty does not apply if the defect of the product resulted from improper or inadequate maintenance, installation, repair, or calibration (performed by a party other than NI); unauthorized modification; improper environment; use of an improper hardware or software key; improper use or operation outside of the specification for the product; improper voltages; accident, abuse, or neglect; or a hazard such as lightning, flood, or other act of nature.

THE REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND THE CUSTOMER’S SOLE REMEDIES, AND SHALL APPLY EVEN IF SUCH REMEDIES FAIL OF THEIR ESSENTIAL PURPOSE.

EXCEPT AS EXPRESSLY SET FORTH HEREIN, PRODUCTS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND AND NI DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, WITH RESPECT TO THE PRODUCTS, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE FROM USAGE OF TRADE OR COURSE OF DEALING. NI DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF OR THE RESULTS OF THE USE OF THE PRODUCTS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NI DOES NOT WARRANT THAT THE OPERATION OF THE PRODUCTS WILL BE UNINTERRUPTED OR ERROR FREE.

In the event that you and NI have a separate signed written agreement with warranty terms covering the products, then the warranty terms in the separate agreement shall control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

End-User License Agreements and Third-Party Legal Notices

You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:

- Notices are located in the `<National Instruments>\_Legal Information` and `<National Instruments>` directories.
- EULAs are located in the `<National Instruments>\Shared\MDF\Legal\license` directory.
- Review `<National Instruments>\_Legal Information.txt` for information on including legal information in installers built with NI products.

U.S. Government Restricted Rights

If you are an agency, department, or other entity of the United States Government (“Government”), the use, duplication, reproduction, release, modification, disclosure or transfer of the technical data included in this manual is governed by the Restricted Rights provisions under Federal Acquisition Regulation 52.227-14 for civilian agencies and Defense Federal Acquisition Regulation Supplement Section 252.227-7014 and 252.227-7015 for military agencies.

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at `ni.com/trademarks` for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and `vernier.com` are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix®, Tek, and Tektronix, Enabling Technology are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

Export Compliance Information

Refer to the *Export Compliance Information* at `ni.com/legal/export-compliance` for the National Instruments global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

YOU ARE ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY AND RELIABILITY OF THE PRODUCTS WHENEVER THE PRODUCTS ARE INCORPORATED IN YOUR SYSTEM OR APPLICATION, INCLUDING THE APPROPRIATE DESIGN, PROCESS, AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

PRODUCTS ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING IN THE OPERATION OF NUCLEAR FACILITIES; AIRCRAFT NAVIGATION; AIR TRAFFIC CONTROL SYSTEMS; LIFE SAVING OR LIFE SUSTAINING SYSTEMS OR SUCH OTHER MEDICAL DEVICES; OR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, “HIGH-RISK USES”). FURTHER, PRUDENT STEPS MUST BE TAKEN TO PROTECT AGAINST FAILURES, INCLUDING PROVIDING BACK-UP AND SHUT-DOWN MECHANISMS. NI EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES.

Contents

About This Manual

Related Documentation xv

Activating Your Software

How Do I Activate My Software? xvii
What is Activation? xvii
What is the NI Activation Wizard? xvii
What Information Do I Need to Activate? xviii
How Do I Find My Product Serial Number? xviii
What is a Computer ID? xviii
How Can I Evaluate NI Software? xix
Moving Software After Activation xix
Deactivating a Product xix
Using Windows Guest Accounts xix

Chapter 1
Introduction

KWP2000 (Key Word Protocol 2000) 1-1
- Transport Protocol 1-2
- Diagnostic Services 1-2
- Diagnostic Service Format 1-2
- Connect/Disconnect 1-3
- GetSeed/Unlock 1-3
- Read/Write Memory 1-3
- Measurements 1-4
- Diagnostic Trouble Codes 1-4
- Input/Output Control 1-4
- Remote Activation of a Routine 1-4
- External References 1-4

UDS (Unified Diagnostic Services) 1-5
- Diagnostic Services 1-5
- Diagnostic Service Format 1-6
- External References 1-6

OBD (On-Board Diagnostic) 1-6

Chapter 2
Installation and Configuration

Installation 2-1
LabVIEW Real-Time (RT) Configuration 2-2
Hardware and Software Requirements 2-3

Chapter 3
Application Development

Choosing the Programming Language 3-1
LabVIEW 3-1
LabWindows/CVI 3-1
Visual C++ 6 3-2
Other Programming Languages 3-3
Application Development on CompactRIO or R Series Using an NI 985*x* or NI 986*x* C Series Module 3-4

Chapter 4
Using the Automotive Diagnostic Command Set

Structure of the Automotive Diagnostic Command Set 4-1
Automotive Diagnostic Command Set API Structure 4-2
General Programming Model 4-3
Available Diagnostic Services 4-4
Tweaking the Transport Protocol 4-4

Chapter 5
Automotive Diagnostic Command Set API for LabVIEW

Section Headings 5-1
Purpose 5-1
Format 5-1
Input and Output 5-1
Description 5-1
List of VIs 5-2
General Functions 5-10
Close Diagnostic.vi 5-10
Convert from Phys.vi 5-12
Convert to Phys.vi 5-14
Create Extended CAN IDs.vi 5-16
Diag Get Property.vi 5-17
Diag Set Property.vi 5-20
Diagnostic Frame Recv.vi 5-23

Diagnostic Frame Send.vi 5-25
Diagnostic Service.vi 5-27
DTC to String.vi 5-29
Get Time Stamp.vi 5-30
OBD Open.vi 5-32
Open Diagnostic.vi 5-36
Open Diagnostic on IP.vi 5-40
Open Diagnostic on LIN.vi 5-42
VWTP Connect.vi 5-45
VWTP Connection Test.vi 5-47
VWTP Disconnect.vi 5-49
DoIP Functions 5-51
DoIP Activate Routing.vi 5-51
DoIP Connect.vi 5-53
DoIP Disconnect.vi 5-55
DoIP Get Diagnostic Power Mode.vi 5-57
DoIP Get DoIP Entity Status.vi 5-59
DoIP Get Entities.vi 5-61
DoIP Send Vehicle Identification Request.vi 5-64
DoIP Send Vehicle Identification Request w EID.vi 5-66
DoIP Send Vehicle Identification Request w VIN.vi 5-68
KWP2000 Services 5-70
ClearDiagnosticInformation.vi 5-70
ControlDTCSetting.vi 5-73
DisableNormalMessageTransmission.vi 5-76
ECUReset.vi 5-78
EnableNormalMessageTransmission.vi 5-80
InputOutputControlByLocalIdentifier.vi 5-82
ReadDataByLocalIdentifier.vi 5-84
ReadDTCByStatus.vi 5-86
ReadECUIdentification.vi 5-89
ReadMemoryByAddress.vi 5-91
ReadStatusOfDTC.vi 5-93
RequestRoutineResultsByLocalIdentifier.vi 5-96
RequestSeed.vi 5-98
SendKey.vi 5-100
StartDiagnosticSession.vi 5-102
StartRoutineByLocalIdentifier.vi 5-104
StopDiagnosticSession.vi 5-106
StopRoutineByLocalIdentifier.vi 5-108
TesterPresent.vi 5-110
WriteDataByLocalIdentifier.vi 5-112
WriteMemoryByAddress.vi 5-114

UDS (DiagOnCAN) Services 5-116
UDS ClearDiagnosticInformation.vi 5-116
UDS CommunicationControl.vi 5-119
UDS ControlDTCSetting.vi 5-121
UDS DiagnosticSessionControl.vi 5-123
UDS ECUReset.vi 5-125
UDS InputOutputControlByIdentifier.vi 5-127
UDS ReadDataByIdentifier.vi 5-129
UDS ReadMemoryByAddress.vi 5-131
UDS ReportDTCBySeverityMaskRecord.vi 5-133
UDS ReportDTCByStatusMask.vi 5-136
UDS ReportSeverityInformationOfDTC.vi 5-139
UDS ReportSupportedDTCs.vi 5-142
UDS RequestDownload.vi 5-145
UDS RequestSeed.vi 5-147
UDS RequestTransferExit.vi 5-149
UDS RequestUpload.vi 5-151
UDS RoutineControl.vi 5-153
UDS SendKey.vi 5-155
UDS TesterPresent.vi 5-157
UDS TransferData.vi 5-159
UDS WriteDataByIdentifier.vi 5-162
UDS WriteMemoryByAddress.vi 5-164
UDS06 ReadMemoryByAddress.vi 5-166
UDS06 WriteMemoryByAddress.vi 5-168
OBD (On-Board Diagnostics) Services 5-170
OBD Clear Emission Related Diagnostic Information.vi 5-170
OBD Request Control Of On-Board Device.vi 5-172
OBD Request Current Powertrain Diagnostic Data.vi 5-174
OBD Request Emission Related DTCs.vi 5-176
OBD Request Emission Related DTCs During Current Drive Cycle.vi 5-179
OBD Request On-Board Monitoring Test Results.vi 5-182
OBD Request Permanent Fault Codes.vi 5-184
OBD Request Powertrain Freeze Frame Data.vi 5-187
OBD Request Supported PIDs.vi 5-189
OBD Request Vehicle Information.vi 5-191
WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) Services 5-193
WWH-OBD Clear Emission Related DTCs.vi 5-193
WWH-OBD Convert DTCs to J1939.vi 5-195
WWH-OBD Convert DTCs to J2012.vi 5-197
WWH-OBD Request DID.vi 5-199
WWH-OBD Request DTC Extended Data Record.vi 5-201
WWH-OBD Request Emission Related DTCs.vi 5-203
WWH-OBD Request Freeze Frame Information.vi 5-206

WWH-OBD Request RID.vi 5-208
WWH-OBD Request Supported DIDs.vi 5-210
WWH-OBD Request Supported RIDs.vi 5-212

Chapter 6
Automotive Diagnostic Command Set API for C

Section Headings 6-1
Purpose 6-1
Format 6-1
Input and Output 6-1
Description 6-1
List of Data Types 6-2
List of Functions 6-3
General Functions 6-13
ndCloseDiagnostic 6-13
ndConvertFromPhys 6-14
ndConvertToPhys 6-16
ndCreateExtendedCANIds 6-18
ndDiagFrameRecv 6-20
ndDiagFrameSend 6-22
ndDiagnosticService 6-23
ndDTCToString 6-25
ndGetProperty 6-26
ndGetTimeStamp 6-29
ndOBDOpen 6-30
ndOpenDiagnostic 6-33
ndOpenDiagnosticOnIP 6-37
ndOpenDiagnosticOnLIN 6-39
ndSetProperty 6-41
ndStatusToString 6-44
ndVWTPConnect 6-46
ndVWTPConnectionTest 6-48
ndVWTPDisconnect 6-49
DoIP Functions 6-50
ndDoIPActivateRouting 6-50
ndDoIPConnect 6-52
ndDoIPDisconnect 6-54
ndDoIPEntityStatus 6-55
ndDoIPGetDiagPowerMode 6-57
ndDoIPGetEntities 6-58
ndDoIPSendVehicleIdentRequest 6-60
ndDoIPSendVehicleIdentReqEID 6-61
ndDoIPSendVehicleIdentReqVIN 6-62

KWP2000 Services 6-63
ndClearDiagnosticInformation 6-63
ndControlDTCSetting 6-65
ndDisableNormalMessageTransmission 6-67
ndECUReset 6-68
ndEnableNormalMessageTransmission 6-70
ndInputOutputControlByLocalIdentifier 6-71
ndReadDataByLocalIdentifier 6-73
ndReadDTCByStatus 6-75
ndReadECUIdentification 6-78
ndReadMemoryByAddress 6-80
ndReadStatusOfDTC 6-82
ndRequestRoutineResultsByLocalIdentifier 6-85
ndRequestSeed 6-87
ndSendKey 6-89
ndStartDiagnosticSession 6-91
ndStartRoutineByLocalIdentifier 6-92
ndStopDiagnosticSession 6-94
ndStopRoutineByLocalIdentifier 6-95
ndTesterPresent 6-97
ndWriteDataByLocalIdentifier 6-99
ndWriteMemoryByAddress 6-101
UDS (DiagOnCAN) Services 6-103
ndUDSClearDiagnosticInformation 6-103
ndUDSCommunicationControl 6-105
ndUDSControlDTCSetting 6-107
ndUDSDiagnosticSessionControl 6-108
ndUDSECUReset 6-109
ndUDSInputOutputControlByIdentifier 6-111
ndUDSReadDataByIdentifier 6-113
ndUDSReadMemoryByAddress 6-115
ndUDSReportDTCBySeverityMaskRecord 6-117
ndUDSReportDTCByStatusMask 6-120
ndUDSReportSeverityInformationOfDTC 6-123
ndUDSReportSupportedDTCs 6-126
ndUDSRequestDownload 6-129
ndUDSRequestSeed 6-131
ndUDSRequestTransferExit 6-133
ndUDSRequestUpload 6-135
ndUDSRoutineControl 6-137
ndUDSSendKey 6-139
ndUDSTesterPresent 6-141
ndUDSTransferData 6-143
ndUDSWriteDataByIdentifier 6-145

ndUDSWriteMemoryByAddress 6-147
ndUDS06ReadMemoryByAddress 6-149
ndUDS06WriteMemoryByAddress 6-151
OBD (On-Board Diagnostics) Services 6-153
ndOBDClearEmissionRelatedDiagnosticInformation 6-153
ndOBDRequestControlOfOnBoardDevice 6-154
ndOBDRequestCurrentPowertrainDiagnosticData 6-156
ndOBDRequestEmissionRelatedDTCs 6-158
ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle 6-160
ndOBDRequestOnBoardMonitoringTestResults 6-162
ndOBDRequestPermanentFaultCodes 6-164
ndOBDRequestPowertrainFreezeFrameData 6-166
ndOBDRequestVehicleInformation 6-168
WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) Services 6-170
ndWWHOBDClearEmissionRelatedDTCs 6-170
ndWWHOBDConvertDTCsToJ1939 6-171
ndWWHOBDConvertDTCsToJ2012 6-173
ndWWHOBDRequestDID 6-175
ndWWHOBDRequestDTCExtendedDataRecord 6-177
ndWWHOBDRequestEmissionRelatedDTCs 6-179
ndWWHOBDRequestFreezeFrameInformation 6-182
ndWWHOBDRequestRID 6-184
ndWWHOBDRequestSupportedDIDs 6-186
ndWWHOBDRequestSupportedRIDs 6-188

Appendix A
NI Services

Index

About This Manual

This manual provides instructions for using the Automotive Diagnostic Command Set. It contains information about installation, configuration, and troubleshooting, and also contains Automotive Diagnostic Command Set function reference for LabVIEW-based and C-based APIs.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *ANSI/ISO Standard 11898-1993, Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*
- *CAN Specification Version 2.0*, 1991, Robert Bosch GmbH., Postfach 106050, D-70049 Stuttgart 1
- *CiA Draft Standard 102, Version 2.0*, CAN Physical Layer for Industrial Applications
- *LIN Specification Package, Revision 2.2*
- *ISO 14230:1999(E), Road Vehicles, Diagnostic Systems, Keyword Protocol 2000*
- *ISO 14229:1998(E), Road Vehicles, Diagnostic Systems, Diagnostic Services Specification*
- *ISO 157651:2004(E), Road Vehicles, Diagnostics on Controller Area Networks (CAN)*
- *ISO 15031-5:2006(E), Road Vehicles, Communication Between Vehicle and External Equipment for Emissions-Related Diagnostics*
- *ISO 27145:2012(E), Road Vehicles, Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) Communication Requirements*
- *NI-CAN Hardware and Software Manual*

Activating Your Software

This section describes how to use the NI Activation Wizard to activate your software.

How Do I Activate My Software?

Use the NI Activation Wizard to obtain an activation code for your software. You can launch the NI Activation Wizard two ways:

- Launch the product and choose to activate your software from the list of options presented.
- Launch NI License Manager by selecting **Start»All Programs» National Instruments»NI License Manager**. Click the **Activate** button in the toolbar.

Notes If your software is a part of a Volume License Agreement (VLA), contact your VLA administrator for installation and activation instructions.

NI software for Mac OS X and Linux operating systems does not require activation.

What is Activation?

Activation is the process of obtaining an activation code to enable your software to run on your computer. An activation code is an alphanumeric string that verifies the software, version, and computer ID to enable features on your computer. Activation codes are unique and are valid on only one computer.

What is the NI Activation Wizard?

The NI Activation Wizard is a part of NI License Manager that steps you through the process of enabling software to run on your machine.

What Information Do I Need to Activate?

You need your product serial number, user name, and organization. The NI Activation Wizard determines the rest of the information. Certain activation methods may require additional information for delivery. This information is used only to activate your product. Complete disclosure of the National Instruments software licensing information privacy policy is available at `ni.com/activate/privacy`. If you optionally choose to register your software, your information is protected under the National Instruments privacy policy, available at `ni.com/privacy`.

How Do I Find My Product Serial Number?

Your serial number uniquely identifies your purchase of NI software. You can find your serial number on the Certificate of Ownership included in your software kit. If your software kit does not include a Certificate of Ownership, you can find your serial number on the product packing slip or on the shipping label.

If you have installed a previous version using your serial number, you can find the serial number by selecting the **Help»About** menu item within the application or by selecting your product within NI License Manager (**Start»All Programs»National Instruments»NI License Manager**). You can also contact your local National Instruments branch.

What is a Computer ID?

The computer ID contains unique information about your computer. National Instruments requires this information to enable your software. You can find your computer ID through the NI Activation Wizard or by using NI License Manager, as follows:

1. Launch NI License Manager by selecting **Start»All Programs» National Instruments»NI License Manager**.
2. Click the **Display Computer Information** button in the toolbar.

For more information about product activation and licensing, refer to `ni.com/activate`.

How Can I Evaluate NI Software?

You can install and run most NI application software in evaluation mode. This mode lets you use a product with certain limitations, such as reduced functionality or limited execution time. Refer to your product documentation for specific information on the product's evaluation mode.

Moving Software After Activation

To transfer your software to another computer, install and activate it on the second computer. You are not prohibited from transferring your software from one computer to another and you do not need to contact or inform NI of the transfer. Because activation codes are unique to each computer, you will need a new activation code. Refer to *How Do I Activate My Software?* to acquire a new activation code and reactivate your software.

Deactivating a Product

To deactivate a product and return the product to the state it was in before you activated it, right-click the product in the NI License Manager tree and select **Deactivate**. If the product was in evaluation mode before you activated it, the properties of the evaluation mode may not be restored.

Using Windows Guest Accounts

NI License Manager does not support Microsoft Windows Guest accounts. You must log in to a non-Guest account to run licensed NI application software.

1 Introduction

Diagnostics involve remote execution of routines, or services, on ECUs. To execute a routine, you send a byte string as a request to an ECU, and the ECU usually answers with a response byte string. Several diagnostic protocols such as KWP2000 and UDS standardize the format of the services to be executed, but those standards leave a large amount of room for manufacturer-specific extensions. A newer trend is the emission-related legislated OnBoard Diagnostics (OBD), which is manufacturer independent and standardized in SAE J1979 and ISO 15031-5. This standard adds another set of services that follow the same scheme.

Because diagnostics were traditionally executed on serial communication links, the byte string length is not limited. For newer, CAN, LIN, or Ethernet-based diagnostics, this involves using a transport protocol that segments the arbitrarily long byte strings into pieces that can be transferred over the CAN or LIN bus, and reassembles them on the receiver side. Several transport protocols accomplish this task. The Automotive Diagnostic Command Set implements the ISO TP (standardized in ISO 15765-2) for CAN and LIN-based diagnostics, the manufacturer-specific VW TP 2.0 for CAN-based diagnostics, and the Diagnostics on IP (DoIP) transport protocol (standardized as ISO 13400) for Ethernet-based diagnostics.

Note The Automotive Diagnostic Command Set is designed for CAN, LIN, or Ethernet-based diagnostics only. Diagnostics on serial lines (K-line and L-line) or FlexRay are not in the scope of the Automotive Diagnostic Command Set.

KWP2000 (Key Word Protocol 2000)

The KWP2000 protocol has become a de facto standard in automotive diagnostic applications. It is standardized as ISO 14230-3. KWP2000 describes the implementation of various diagnostic services you can access through the protocol. You can run KWP2000 on several transport layers such as K-line (serial) or CAN.

Transport Protocol

As KWP2000 uses messages of variable byte lengths, a transport protocol is necessary on layers with only a well defined (short) message length, such as CAN. The transport protocol splits a long KWP2000 message into pieces that can be transferred over the network and reassembles those pieces to recover the original message.

KWP2000 runs on CAN on various transport protocols such as ISO TP (ISO 15765-2), TP 1.6, TP 2.0 (Volkswagen), SAE J1939-21, and Diagnostic Over IP (ISO 13400).

Note For KWP2000, the Automotive Diagnostic Command Set supports only the ISO TP (standardized in ISO 15765-2), manufacturer-specific VW TP 2.0 transport protocols, and Diagnostic Over IP (ISO 13400).

Diagnostic Services

The diagnostic services available in KWP2000 are grouped in functional units and identified by a one-byte code (ServiceId). The standard does not define all codes; for some codes, the standard refers to other SAE or ISO standards, and some are reserved for manufacturer-specific extensions. The Automotive Diagnostic Command Set supports the following services:

- Diagnostic Management
- Data Transmission
- Stored Data Transmission (Diagnostic Trouble Codes)
- Input/Output Control
- Remote Activation of Routine

Note Upload/Download and Extended services are not part of the Automotive Diagnostic Command Set.

Diagnostic Service Format

Diagnostic services have a common message format. Each service defines a Request Message, Positive Response Message, and Negative Response Message.

The Request Message has the ServiceId as first byte, plus additional service-defined parameters. The Positive Response Message has an echo of the ServiceId with bit 6 set as first byte, plus the service-defined response parameters.

The Negative Response Message is usually a three-byte message: it has the Negative Response ServiceId as first byte, an echo of the original ServiceId as second byte, and a ResponseCode as third byte. The only exception to this format is the negative response to an EscapeCode service; here, the third byte is an echo of the user-defined service code, and the fourth byte is the ResponseCode. The KWP2000 standard partly defines the ResponseCodes, but there is room left for manufacturer-specific extensions. For some of the ResponseCodes, KWP2000 defines an error handling procedure. Because both positive and negative responses have an echo of the requested service, you can always assign the responses to their corresponding request.

Connect/Disconnect

KWP2000 expects a diagnostic session to be started with StartDiagnosticSession and terminated with StopDiagnosticSession. However, StartDiagnosticSession has a DiagnosticMode parameter that determines the diagnostic session type. Depending on this type, the ECU may or may not support other diagnostic services, or operate in a restricted mode where not all ECU functions are available. The DiagnosticMode parameter values are manufacturer specific and not defined in the standard.

For a diagnostic session to remain active, it must execute the TesterPresent service periodically if no other service is executed. If the TesterPresent service is missing for a certain period of time, the diagnostic session is terminated, and the ECU returns to normal operation mode.

GetSeed/Unlock

A GetSeed/Unlock mechanism may protect some diagnostic services. However, the applicable services are left to the manufacturer and not defined by the standard.

You can execute the GetSeed/Unlock mechanism through the SecurityAccess service. This defines several levels of security, but the manufacturer assigns these levels to certain services.

Read/Write Memory

Use the Read/WriteMemoryByAddress services to upload/download data to certain memory addresses on an ECU. The address is a three-byte quantity in KWP2000 and a five-byte quantity (four-byte address and one-byte extension) in the calibration protocols.

The Upload/Download functional unit services are highly manufacturer specific and not well defined in the standard, so they are not a good way to provide a general upload/download mechanism.

Measurements

Use the ReadDataByLocal/CommonIdentifier services to access ECU data in a way similar to a DAQ list. A Local/CommonIdentifier describes a list of ECU quantities that are then transferred from the ECU to the tester. The transfer can be either single value or periodic, with a slow, medium, or fast transfer rate. The transfer rates are manufacturer specific; you can use the SetDataRates service to set them, but this setting is manufacturer specific.

Note The Automotive Diagnostic Command Set supports single-point measurements.

Diagnostic Trouble Codes

A major diagnostic feature is the readout of Diagnostic Trouble Codes (DTCs). KWP2000 defines several services that access DTCs based on their group or status.

Input/Output Control

KWP2000 defines services to modify internal or external ECU signals. One example is redirecting ECU sensor inputs to stimulated signals. The control parameters of these commands are manufacturer specific and not defined in the standard.

Remote Activation of a Routine

These services are similar to the ActionService and DiagService functions of CCP. You can invoke an ECU internal routine identified by a Local/CommonIdentifier or a memory address. Contrary to the CCP case, execution of this routine can be asynchronous; that is, there are separate Start, Stop, and RequestResult services.

The control parameters of these commands are manufacturer specific and not defined in the standard.

External References

For more information about the KWP2000 Standard, refer to the ISO 14230-3 standard.

UDS (Unified Diagnostic Services)

The UDS protocol has become a de facto standard in automotive diagnostic applications. It is standardized as ISO 14229. UDS describes the implementation of various diagnostic services you can access through the protocol.

As UDS uses messages of variable byte lengths, a transport protocol is necessary on layers with only a well defined (short) message length, such as CAN or LIN. The transport protocol splits a long UDS message into pieces that can be transferred over the network and reassembles those pieces to recover the original message.

UDS runs on CAN, LIN, and Ethernet on various transport protocols.

Note The Automotive Diagnostic Command Set supports only the ISO TP (standardized in ISO 15765-2), manufacturer-specific VW TP 2.0 transport protocols, and Diagnostic Over IP (ISO 13400).

Diagnostic Services

The diagnostic services available in UDS are grouped in functional units and identified by a one-byte code (ServiceId). Not all codes are defined in the standard; for some codes, the standard refers to other standards, and some are reserved for manufacturer-specific extensions. The Automotive Diagnostic Command Set supports the following services:

- Diagnostic Management
- Data Transmission
- Stored Data Transmission (Diagnostic Trouble Codes)
- Input/Output Control
- Remote Activation of Routine

For UDS on LIN, a slave node must support a set of ISO 14229-1 diagnostic services such as:

- Node identification (reading hardware and software version, hardware part number, and diagnostic version)
- Reading data parameters (reading ECU internal values such as oil temperature and vehicle speed)
- Writing parameter values if applicable

Note For more information about the LIN Diagnostic service implementations, refer to the *LIN Specification Package,* Revision 2.2, from the LIN Consortium.

Diagnostic Service Format

Diagnostic services have a common message format. Each service defines a Request Message, a Positive Response Message, and a Negative Response Message. The general format of the diagnostic services complies with the KWP2000 definition; most of the Service Ids also comply with KWP2000. The Request Message has the ServiceId as first byte, plus additional service-defined parameters. The Positive Response Message has an echo of the ServiceId with bit 6 set as first byte, plus the service-defined response parameters.

Note Some parameters to both the Request and Positive Response Messages are optional. Each service defines these parameters. Also, the standard does not define all parameters.

The Negative Response Message is usually a three-byte message: it has the Negative Response ServiceId (0x7F) as first byte, an echo of the original ServiceId as second byte, and a ResponseCode as third byte. The UDS standard partly defines the ResponseCodes, but there is room left for manufacturer-specific extensions. For some of the ResponseCodes, UDS defines an error handling procedure.

Because both positive and negative responses have an echo of the requested service, you always can assign the responses to their corresponding request.

External References

For more information about the UDS Standard, refer to the ISO 15765-3 standard.

OBD (On-Board Diagnostic)

On-Board Diagnostic (OBD) systems are present in most cars and light trucks on the road today. On-Board Diagnostics refer to the vehicle's self-diagnostic and reporting capability, which the vehicle owner or a repair technician can use to query status information for various vehicle subsystems.

The amount of diagnostic information available via OBD has increased since the introduction of on-board vehicle computers in the early 1980s. Modern OBD implementations use a CAN communication port to provide real-time data and a standardized series of diagnostic trouble codes

(DTCs), which identify and remedy malfunctions within the vehicle. In the 1970s and early 1980s, manufacturers began using electronic means to control engine functions and diagnose engine problems. This was primarily to meet EPA emission standards. Through the years, on-board diagnostic systems have become more sophisticated. OBD-II, a standard introduced in the mid 1990s, provides almost complete engine control and also monitors parts of the chassis, body, and accessory devices, as well as the car's diagnostic control network. The newest standard was introduced in 2012 as WWH-OBD.

The On-Board Diagnostic (OBD) standard defines a minimum set of diagnostic information for passenger cars and light and medium-duty trucks, which must be exchanged with any off-board test equipment.

2

Installation and Configuration

This chapter explains how to install and configure the Automotive Diagnostic Command Set.

Installation

This section discusses the Automotive Diagnostic Command Set installation for Microsoft Windows.

Note You need administrator rights to install the Automotive Diagnostic Command Set on your computer.

Follow these steps to install the Automotive Diagnostic Command Set software:

1. Insert the Automotive Diagnostic Command Set CD into the CD-ROM drive.
2. Open Windows Explorer.
3. Access the CD-ROM drive.
4. Double-click on `autorun.exe` to launch the software interface.
5. Start the installation. The installation program guides you through the rest of the installation process.
6. If you have not already installed NI-CAN, the Automotive Diagnostic Command Set installer automatically installs the NI-CAN driver on your computer.

   Within the **Devices & Interfaces** branch of the MAX Configuration tree, NI CAN hardware is listed along with other hardware in the local computer system. If the CAN hardware is not listed here, MAX is not configured to search for new devices on startup. To search for the new hardware, press <F5>. To verify installation of the CAN hardware, right-click the CAN device, then select **Self-test**. If the self-test passes, the card icon shows a checkmark. If the self-test fails, the card icon shows an X mark, and the **Test Status** in the right pane describes the problem.

Refer to Appendix A, *Troubleshooting and Common Questions*, of the *NI-CAN User Manual* for information about resolving hardware installation problems.

If you are using the Automotive Diagnostic Command Set on an NI-XNET device, install the NI-XNET driver 1.0 or higher, NI-CAN 2.7 or higher, and the NI-CAN Compatibility Library on your computer.

The MAX Configuration tree **Devices and Interfaces** branch lists NI-XNET hardware (along with other local computer system hardware). If the NI-XNET hardware is not listed there, MAX is not configured to search for new devices on startup. To search for the new hardware, press <F5>. To verify CAN hardware installation, right-click the CAN device and select **Self-Test**. If the self-test passes, the card icon shows a checkmark. If the self-test fails, the card icon shows an X mark, and the **Test Status** in the right pane describes the problem. Refer to Chapter 6, *Troubleshooting and Common Questions,* of the *NI-XNET User Manual* for information about resolving hardware installation problems. The NI-XNET CAN hardware interfaces are listed under the device name. To change the interface name, select a new one from the **Interface Name** box in the middle pane.

When installation is complete, you can access the Automotive Diagnostic Command Set functions in your application development environment.

LabVIEW Real-Time (RT) Configuration

LabVIEW Real-Time (RT) combines easy-to-use LabVIEW programming with the power of real-time systems. When you use a National Instruments PXI controller as a LabVIEW RT system, you can install a PXI CAN card and use the NI-CAN or NI-XNET APIs to develop real-time applications. As with any NI software library for LabVIEW RT, you must install the Automotive Diagnostic Command Set software to the LabVIEW RT target using the Remote Systems branch in MAX. For more information, refer to the LabVIEW RT documentation.

After you install the PXI CAN cards and download the Automotive Diagnostic Command Set software to the LabVIEW RT system, you must verify the installation.

Hardware and Software Requirements

You can use the Automotive Diagnostic Command Set on the following hardware:

- National Instruments NI-CAN hardware Series 1 or 2 with the NI-CAN driver software version 2.3 or later installed.
- National Instruments NI-XNET hardware with the NI-XNET driver software version 1.0 or later installed.
- National Instruments CompactRIO or R Series Multifunction RIO hardware and the NI 9853 or NI 9852 CompactRIO CAN modules.

Note You can use the Automotive Diagnostic Command Set with LabVIEW 2009 or newer on CompactRIO systems or National Instruments R Series Multifunction RIO hardware.

3

Application Development

This chapter explains how to develop an application using the Automotive Diagnostic Command Set API.

Choosing the Programming Language

The programming language you use for application development determines how to access the Automotive Diagnostic Command Set APIs.

LabVIEW

Automotive Diagnostic Command Set functions and controls are in the LabVIEW palettes. In LabVIEW, the Automotive Diagnostic Command Set palette is in the top-level NI Measurements palette.

Chapter 5, *Automotive Diagnostic Command Set API for LabVIEW*, describes each LabVIEW VI for the Automotive Diagnostic Command Set API.

To access the VI reference from within LabVIEW, press <Ctrl-H> to open the Help window, click the appropriate Automotive Diagnostic Command Set VI, and follow the link. The Automotive Diagnostic Command Set software includes a full set of LabVIEW examples. These examples teach programming basics as well as advanced topics. The example help describes each example and includes a link you can use to open the VI.

LabWindows/CVI

Within LabWindows™/CVI™, the Automotive Diagnostic Command Set function panel is in **Libraries»Automotive Diagnostic Command Set**. As with other LabWindows/CVI function panels, the Automotive Diagnostic Command Set function panel provides help for each function and the ability to generate code. Chapter 6, *Automotive Diagnostic Command Set API for C*, describes each Automotive Diagnostic Command Set API function. You can access the reference for each function directly from within the function panel. The Automotive Diagnostic Command Set API header file is `nidiagcs.h`. The Automotive Diagnostic Command Set API library is `nidiagcs.lib`. The toolkit software includes a full set of

LabWindows/CVI examples. The examples are in the LabWindows/CVI `\samples\Automotive Diagnostic Command Set` directory. Each example includes a complete LabWindows/CVI project (`.prj` file). The example description is in comments at the top of the `.c` file.

Visual C++ 6

The Automotive Diagnostic Command Set software supports Microsoft Visual C/C++ 6.

The header file for Visual C/C++ 6 is in the `Program Files\National Instruments\Shared\ExternalCompilerSupport\C\include` folder. To use the Automotive Diagnostic Command Set API, include the `nidiagcs.h` header file in the code, then link with the `nidiagcs.lib` library file. The library file is in the `Program Files\National Instruments\Shared\ExternalCompilerSupport\C\lib32\msvc` folder.

For C applications (files with a `.c` extension), include the header file by adding a `#include` to the beginning of the code, as follows:

```
#include "nidiagcs.h"
```

For C++ applications (files with a `.cpp` extension), define `__cplusplus` before including the header, as follows:

```
#define __cplusplus
#include "nidiagcs.h"
```

The `__cplusplus` define enables the transition from C++ to the C language functions.

Chapter 6, *Automotive Diagnostic Command Set API for C*, describes each function.

On Windows Vista (with Standard User Account), the typical path to the C examples folder is `\Users\Public\Documents\National Instruments\Automotive Diagnostic Command Set\ Examples\MS Visual C`.

On Windows XP/2000, the typical path to the C examples folder is `\Documents and Settings\All Users\Documents\National Instruments\Automotive Diagnostic Command Set\ Examples\MS Visual C`.

Each example is in a separate folder. The example description is in comments at the top of the `.c` file. At the command prompt, after setting MSVC environment variables (such as with MS `vcvars32.bat`), you can build each example using a command such as:

```
cl /I<HDir> GetDTCs.c <LibDir>\nidiagcs.lib
```

<HDir> is the folder where `nidiagcs.h` can be found.

<LibDir> is the folder where `nidiagcs.lib` can be found.

Other Programming Languages

The Automotive Diagnostic Command Set software does not provide formal support for programming languages other than those described in the preceding sections. If the programming language includes a mechanism to call a Dynamic Link Library (DLL), you can create code to call Automotive Diagnostic Command Set functions. All functions for the Automotive Diagnostic Command Set API are in `nidiagcs.dll`. If the programming language supports the Microsoft Win32 APIs, you can load pointers to Automotive Diagnostic Command Set functions in the application. The following section describes how to use the Win32 functions for C/C++ environments other than Visual C/C++ 6. For more detailed information, refer to Microsoft documentation.

The following C language code fragment shows how to call Win32 `LoadLibrary` to load the Automotive Diagnostic Command Set API DLL:

```
#include <windows.h>
#include "nidiagcs.h"
HINSTANCE NiDiagCSLib = NULL;
NiMcLib = LoadLibrary("nidiagcs.dll");
```

Next, the application must call the Win32 `GetProcAddress` function to obtain a pointer to each Automotive Diagnostic Command Set function the application uses. For each function, you must declare a pointer variable using the prototype of the function. For the Automotive Diagnostic Command Set function prototypes, refer to Chapter 6, *Automotive Diagnostic Command Set API for C*. Before exiting the application, you must unload the Automotive Diagnostic Command Set DLL as follows:

```
FreeLibrary (NiDiagCSLib);
```

Application Development on CompactRIO or R Series Using an NI 985*x* or NI 986*x* C Series Module

To run a project on an FPGA target with an NI 985*x* C Series module, you need an FPGA bitfile (`.lvbitx`). The FPGA bitfile is downloaded to the FPGA target on the execution host. A bitfile is a compiled version of an FPGA VI. FPGA VIs, and thus bitfiles, define the CAN, analog, digital, and pulse width modulation (PWM) inputs and outputs of an FPGA target. The Automotive Diagnostic Command Set does not include FPGA bitfiles for any FPGA target. Refer to the LabVIEW FPGA Module documentation for more information about creating FPGA VIs and bitfiles for an FPGA target.

The default FPGA VI is sufficient for a basic Automotive Diagnostic Command Set application. However, in some situations you may need to modify the existing FPGA code to create a custom bitfile. For example, to use additional I/O on the FPGA target, you must add these I/O to the FPGA VI. You must install the LabVIEW FPGA Module to create these files.

Modify the FPGA VI according to the following guidelines:

- Do not modify, remove, or rename any block diagram controls and indicators named __CAN0 Rx Data, __CAN0 Rx Ready, __CAN0 Tx Data Frame, __CAN0 Tx Ready, __CAN0 Bit Timing, __CAN0 FPGA Is Running, __CAN0 Start, __CAN0 FIFO Full, or __CAN0 FIFO Empty. If you intend to use multiple CAN 985*x* modules on your FPGA, you need to duplicate and rename all controls and indicators accordingly.
- Do not modify the CAN read and write code except to filter CAN IDs on the receiving side to minimize the amount of CAN data transfers to the host.
- As you create controls or indicators, ensure that each control name is unique within the VI.

Refer to the LabVIEW FPGA Module documentation for more information about creating FPGA VIs and bitfiles for an FPGA target.

When using ADCS on CompactRIO with an NI 985*x* C Series module, the interface name is based on the bitfile you use and the interface name you set. For example, `MyInterface@MyBitfile.lvbitx`, `CAN@lvbitfile.lvbitx`, or `CAN0@mybitfile.lvbitx`.

The interface name you use must be part of all parameters in the FPGA code for the CAN communication. Also, the ADCS needs the interface name for correct functionality.

If you define the interface name to be *CAN0*, you must name the parameters as follows:

- __CAN0 Rx Data
- __CAN0 Rx Ready
- __CAN0 Tx Data Frame
- __CAN0 Tx Ready
- __CAN0 Bit Timing
- __CAN0 FPGA Is Running
- __CAN0 Start
- __CAN0 FIFO Full
- __CAN0 FIFO Ready

In addition, you need to set the name of the internally used FIFO to __*CAN0 FIFO* (the FIFO is set to U32, 1029 elements, target scoped, and block memory).

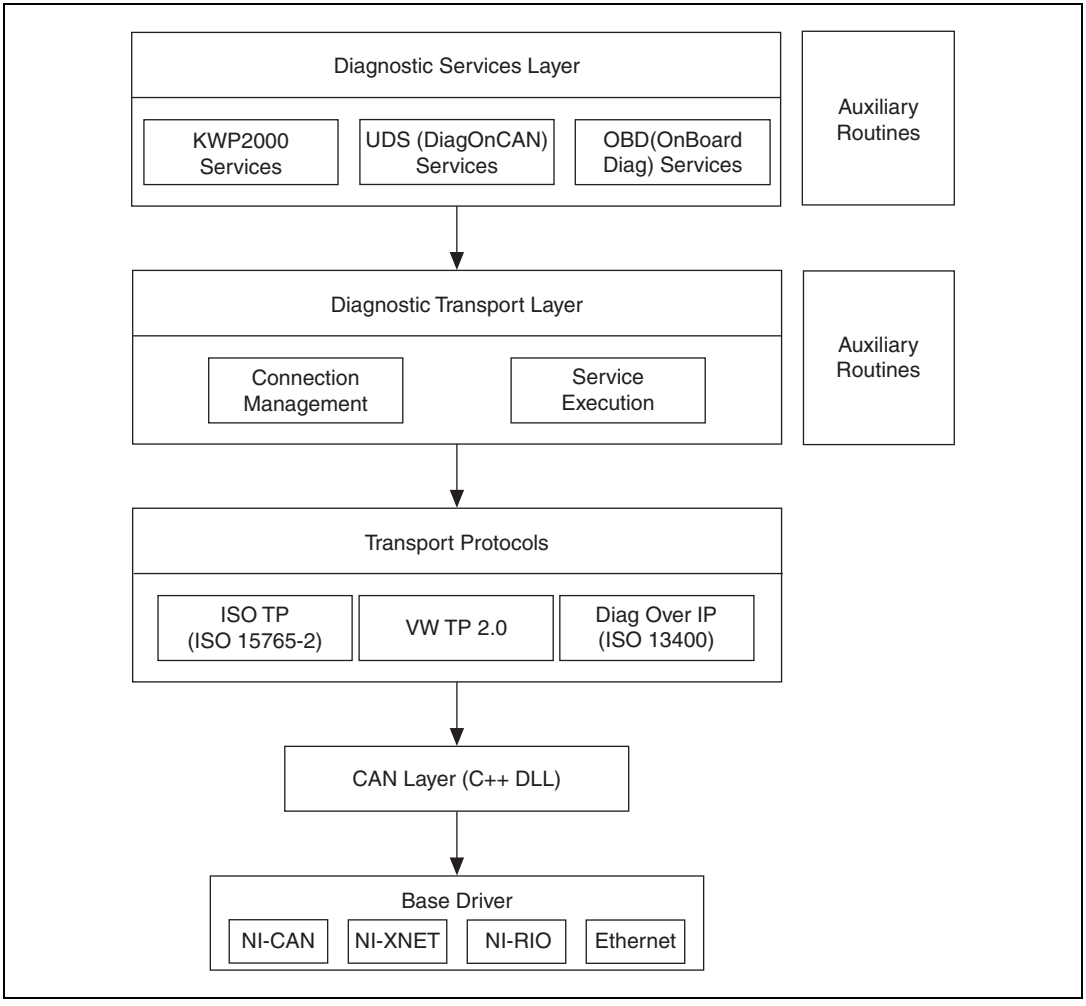
After recompiling your FPGA VI, copy the bitfile to the root directory of your CompactRIO controller and specify the bitfile in the interface name. Or copy the file to any location on the CompactRIO controller and specify an absolute path or path relative to the root for the bitfile.

If you are using an NI-XNET 986*x* C Series module on your CompactRIO target, you need to start an FPGA VI on the target before accessing the port with ADCS. Refer to the *Getting Started with CompactRIO* section in the *NI-XNET Hardware and Software Manual* for more information about compiling the FPGA VI. When the VI is running, you can access the NI 986*x* module as you would program on a Windows or PXI LabVIEW Real-Time target.

4

Using the Automotive Diagnostic Command Set

Structure of the Automotive Diagnostic Command Set



The Automotive Diagnostic Command Set is structured into three layers of functionality:

- The top layer implements three sets of diagnostic services for the diagnostic protocols KWP2000, UDS (DiagOnCAN), and OBD (On-Board Diagnostics).
- The second layer implements general routines involving opening and closing diagnostic communication connections, connecting and disconnecting to/from an ECU, and executing a diagnostic service on byte level. The latter routine is the one the top layer uses heavily.
- The third layer implements the transport protocols needed for diagnostic communication to an ECU. The second layer uses these routines to communicate to an ECU.

All three top layers are fully implemented in LabVIEW.

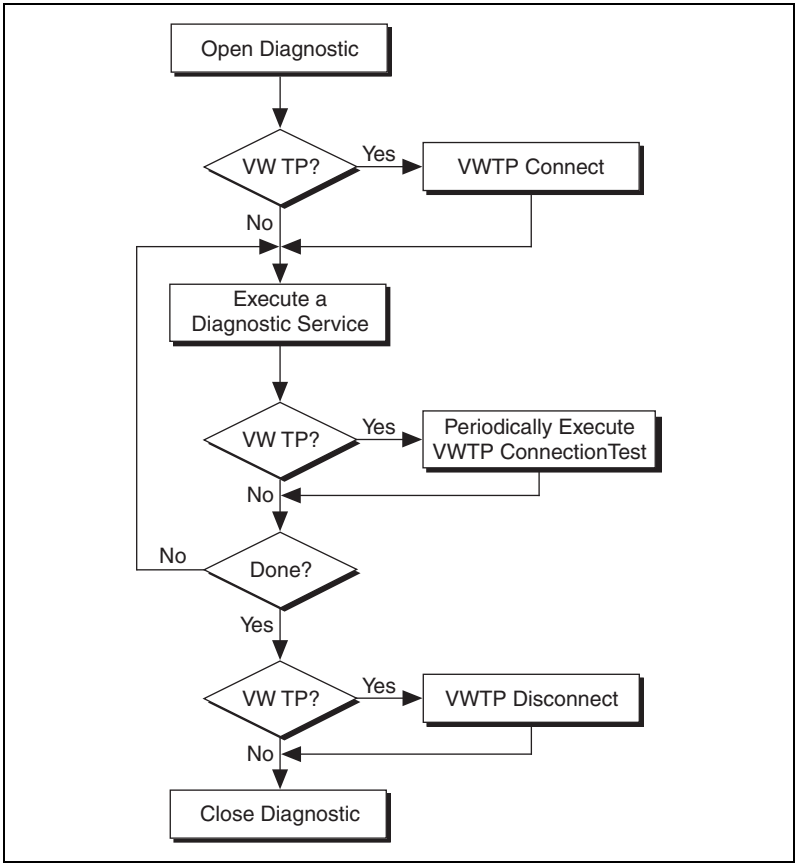
The transport protocols then execute CAN/LIN Read/Write operations through a specialized DLL for streamlining the CAN/LIN data flow, especially in higher busload situations.

Automotive Diagnostic Command Set API Structure

The top two layer routines are available as API functions. Each diagnostic service for KWP2000, UDS, and OBD is available as one routine. Also available on the top level are auxiliary routines for converting scaled physical data values to and from their binary representations used in the diagnostic services.

On the second layer are more general routines for opening and closing diagnostic communication channels and executing a diagnostic service. Auxiliary routines create the diagnostic identifiers from the logical ECU address.

General Programming Model



First, you must open a diagnostic communication link. This involves initializing the CAN/LIN port and defining communication parameters such as the baud rate. For CAN-based diagnostics, the CAN identifiers on which the diagnostic communication takes place must be defined also. No actual communication to the ECU takes place at this stage.

For the VW TP 2.0, you then must establish a communication channel to the ECU using the VWTP Connect routine. The communication channel properties are negotiated between the host and ECU.

After these steps, the diagnostic communication is established, and you can execute diagnostic services of your choice. Note that for the VW TP 2.0,

you must execute the VWTP ConnectionTest routine periodically (once per second) to keep the communication channel open.

When you finish your diagnostic services, you must close the diagnostic communication link. This finally closes the CAN or LIN port. For the VW TP 2.0, you should disconnect the communication channel established before closing.

Available Diagnostic Services

The standards on automotive diagnostic define many different services for many purposes. Unfortunately, most services leave a large amount of room for manufacturer-specific variants and extensions. National Instruments implemented the most used variants while trying not to overload them with optional parameters.

However, all services are implemented in LabVIEW and open to the user. If you are missing a service or variant of an existing service, you can easily add or modify it on your own.

In the C API, you can also implement your own diagnostic services using the ndDiagnosticService routine. However, the templates from the existing services are not available.

Tweaking the Transport Protocol

A set of global constants controls transport protocol behavior. These constants default to maximum performance. To check the properties of an implementation of a transport protocol in an ECU, for example, you may want to change the constants to nonstandard values using the Get/Set Property routines.

5

Automotive Diagnostic Command Set API for LabVIEW

This chapter lists the LabVIEW VIs for the Automotive Diagnostic Command Set API and describes the format, purpose, and parameters for each VI. The VIs are listed alphabetically in four categories: general functions, KWP2000 services, UDS (DiagOnCAN) services, and OBD (On-Board Diagnostics) services.

Section Headings

The following are section headings found in the Automotive Diagnostic Command Set API for LabVIEW VIs.

Purpose

Each VI description briefly describes the VI purpose.

Format

The format section describes the VI format.

Input and Output

The input and output sections list the VI parameters.

Description

The description section gives details about the VI purpose and effect.

List of VIs

The following table is an alphabetical list of the Automotive Diagnostic Command Set VIs.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW

Function	Purpose
ClearDiagnosticInformation.vi	Executes the ClearDiagnosticInformation service and clears selected Diagnostic Trouble Codes (DTCs).
Close Diagnostic.vi	Closes a diagnostic session.
ControlDTCSetting.vi	Executes the ControlDTCSetting service and modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).
Convert from Phys.vi	Converts a physical data value into a binary representation using a type descriptor.
Convert to Phys.vi	Converts a binary representation of a value into its physical value using a type descriptor.
Create Extended CAN IDs.vi	Creates diagnostic CAN IDs according to ISO 15765-2.
Diag Get Property.vi	Gets a diagnostic global internal parameter.
Diag Set Property.vi	Sets a diagnostic global internal parameter.
Diagnostic Frame Recv.vi	Receives a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.
Diagnostic Frame Send.vi	Sends a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.
Diagnostic Service.vi	Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this VI.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
DisableNormalMessageTransmission.vi	Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).
DoIP Activate Routing.vi	Defines the source and target addresses for a DoIP TCP/IP connection.
DoIP Connect.vi	Creates a TCP/IP connection to a DoIP entity identified by its IP address.
DoIP Disconnect.vi	Disconnects the TCP/IP connection to a DoIP entity.
DoIP Get Diagnostic Power Mode.vi	Gets information sbout the DoIP entity power state.
DoIP Get DoIP Entity Status.vi	Gets status information from a DoIP entity.
DoIP Get Entities.vi	Returns a table of all DoIP entities (vehicles) on the local subnet, possibly restricted to EID or VIN.
DoIP Send Vehicle Identification Request.vi	Sends a UDP request to all DoIP-capable vehicles in the local subnet to identify themselves.
DoIP Send Vehicle Identification Request w EID.vi	Sends a UDP request to all DoIP-capable vehicles with a certain EID (MAC address) in the local subnet to identify themselves.
DoIP Send Vehicle Identification Request w VIN.vi	Sends a UDP request to all DoIP-capable vehicles with a certain VIN (Vehicle Identification Number) in the local subnet to identify themselves.
DTC to String.vi	Returns a string representation (such as *P1234*) for a 2-byte Diagnostic Trouble Code (DTC).
Get Time Stamp.vi	Gets timestamp information about the first/last send/received frame of the ISO TP for CAN and LIN.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
ECUReset.vi	Executes the ECUReset service and resets the ECU.
EnableNormalMessageTransmission.vi	Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).
InputOutputControlByLocalIdentifier.vi	Executes the InputOutputControlByLocalIdentifier service. Modifies the ECU I/O port behavior.
OBD Clear Emission Related Diagnostic Information.vi	Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related Diagnostic Trouble Codes (DTCs) in the ECU.
Get Time Stamp.vi	Opens an OBD-II diagnostic session on a CAN port.
OBD Request Control Of On-Board Device.vi	Executes the OBD Request Control Of On-Board Device service. Use this VI to modify ECU I/O port behavior.
OBD Request Current Powertrain Diagnostic Data.vi	Executes the OBD Request Current Powertrain Diagnostic Data service. Reads a data record from the ECU.
OBD Request Emission Related DTCs.vi	Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).
OBD Request Emission Related DTCs During Current Drive Cycle.vi	Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.
OBD Request On-Board Monitoring Test Results.vi	Executes the OBD Request On-Board Monitoring Test Results service. Reads a test data record from the ECU.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
OBD Request Permanent Fault Codes.vi	Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.
OBD Request Powertrain Freeze Frame Data.vi	Executes the OBD Request Powertrain Freeze Frame Data service. Reads a data record from the ECU that has been stored while a Diagnostic Trouble Code occurred.
OBD Request Supported PIDs.vi	Executes the OBD Request Current Powertrain Diagnostic Data service to retrieve the valid PID values for this service.
OBD Request Vehicle Information.vi	Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.
Open Diagnostic.vi	Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.
Open Diagnostic on IP.vi	Opens a diagnostic session on an IP port. Communication to the ECU is not yet started.
Open Diagnostic on LIN.vi	Opens a diagnostic session on an NI-XNET LIN port. Communication to the ECU is not yet started.
ReadDataByLocalIdentifier.vi	Executes the ReadDataByLocalIdentifier service. Reads a data record from the ECU.
ReadDTCByStatus.vi	Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).
ReadECUIdentification.vi	Executes the ReadECUIdentification service. Returns ECU identification data from the ECU.
ReadMemoryByAddress.vi	Executes the ReadMemoryByAddress service. Reads data from the ECU memory.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
ReadStatusOfDTC.vi	Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).
RequestRoutineResultsByLocalIdentifier.vi	Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from a routine on the ECU.
RequestSeed.vi	Executes the SecurityAccess service to retrieve a seed from the ECU.
SendKey.vi	Executes the SecurityAccess service to send a key to the ECU.
StartDiagnosticSession.vi	Executes the StartDiagnosticSession service. Sets up the ECU in a specific diagnostic mode.
StartRoutineByLocalIdentifier.vi	Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.
StopDiagnosticSession.vi	Executes the StopDiagnosticSession service. Brings the ECU back in normal mode.
StopRoutineByLocalIdentifier.vi	Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.
TesterPresent.vi	Executes the TesterPresent service. Keeps the ECU in diagnostic mode.
UDS ClearDiagnosticInformation.vi	Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).
UDS CommunicationControl.vi	Executes the UDS CommunicationControl service. Use this VI to switch on or off transmission and/or reception of the normal communication messages (usually CAN messages).
UDS ControlDTCSetting.vi	Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) generation behavior.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
UDS DiagnosticSessionControl.vi	Executes the UDS DiagnosticSessionControl service. Sets up the ECU in a specific diagnostic mode.
UDS ECUReset.vi	Executes the UDS ECUReset service. Resets the ECU.
UDS InputOutputControlByIdentifier.vi	Executes the UDS InputOutputControlByIdentifier service. Use this VI to modify ECU I/O port behavior.
UDS ReadDataByIdentifier.vi	Executes the UDS ReadDataByIdentifier service. Reads a data record from the ECU.
UDS ReadMemoryByAddress.vi	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.
UDS ReportDTCBySeverityMaskRecord.vi	Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportDTCByStatusMask.vi	Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportSeverityInformationOfDTC.vi	Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
UDS ReportSupportedDTCs.vi	Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
UDS RequestDownload.vi	Initiates a download of data to the ECU.
UDS RequestSeed.vi	Executes the UDS SecurityAccess service to retrieve a seed from the ECU.
UDS RequestTransferExit.vi	Terminates a download/upload process.
UDS RequestUpload.vi	Initiates an upload of data from the ECU.
UDS RoutineControl.vi	Executes the UDS RoutineControl service. Executes a routine on the ECU.
UDS SendKey.vi	Executes the SecurityAccess service to send a key to the ECU.
UDS TesterPresent.vi	Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.
UDS TransferData.vi	Transfers data to/from the ECU in a download/upload process.
UDS WriteDataByIdentifier.vi	Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.
UDS WriteMemoryByAddress.vi	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
UDS06 ReadMemoryByAddress.vi	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.
UDS06 WriteMemoryByAddress.vi	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
VWTP Connect.vi	Establishes a connection channel to an ECU using the VW TP 2.0.
VWTP Connection Test.vi	Maintains a connection channel to an ECU using the VW TP 2.0.
VWTP Disconnect.vi	Terminates a connection channel to an ECU using the VW TP 2.0.
WriteDataByLocalIdentifier.vi	Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.
WriteMemoryByAddress.vi	Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

Table 5-1. Automotive Diagnostic Command Set API VIs for LabVIEW (Continued)

Function	Purpose
WWH-OBD Clear Emission Related DTCs.vi	Executes the WWH-OBD ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).
WWH-OBD Convert DTCs to J1939.vi	Converts DTCs to the J1939 DTC format.
WWH-OBD Convert DTCs to J2012.vi	Converts DTCs to the J2012 DTC format.
WWH-OBD Request DID.vi	Executes the WWH-OBD ReadDataByIdentifier service. Reads a data record from the ECU.
WWH-OBD Request DTC Extended Data Record.vi	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
WWH-OBD Request Emission Related DTCs.vi	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
WWH-OBD Request Freeze Frame Information.vi	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
WWH-OBD Request RID.vi	Executes the WWH-OBD RoutineControl service. Reads a data record from the ECU.
WWH-OBD Request Supported DIDs.vi	Executes the WWH-OBD ReadDataByIdentifier service to retrieve the valid DID values for this service.
WWH-OBD Request Supported RIDs.vi	Executes the WWH-OBD RoutineControl service to retrieve the valid RID values for this service.

General Functions

Close Diagnostic.vi

Purpose

Closes a diagnostic session.

Format

Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

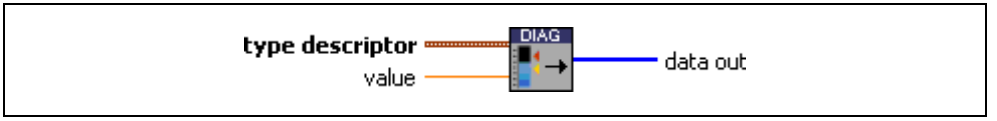
The diagnostic session specified by **diag ref in** is closed, and you can no longer use it for communication to an ECU. Note that this command does not communicate the closing to the ECU before terminating; if this is necessary, you must manually do so (for example, by calling **StopDiagnosticSession.vi**) before calling **Close Diagnostic.vi**.

Convert from Phys.vi

Purpose

Converts a physical data value into a binary representation using a type descriptor.

Format



Input

type descriptor is a cluster that specifies the conversion of the physical value to its binary representation:

Start Byte gives the start byte of the binary representation. For **Convert from Phys.vi**, this value is ignored and always assumed to be 0.

Byte Length is the binary representation byte length.

Byte Order is the byte ordering of the data in the binary representation:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

Data Type is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths of 4 or 8 are allowed.

Scale Factor defines the physical value scaling:

Phys = (Scale Factor) * (binary representation) + (Scale Offset)

Scale Offset (refer to **Scale Factor**)

value is the physical value to be converted.

Output

data out is the binary representation of the physical value. If you build a record of multiple values, you can concatenate the outputs of several instances of **Convert from Phys.vi**.

Description

Data input to diagnostic services (for example, **WriteDataByLocalIdentifier.vi**) is usually a byte stream of binary data. If you have a description of the data input (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use **Convert from Phys.vi** to convert the physical value to the byte stream by filling an appropriate type descriptor cluster.

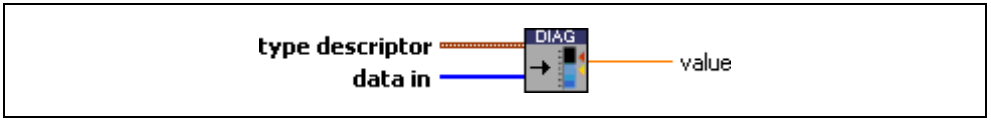
Convert from Phys.vi converts only the portion specified by one type descriptor to a binary representation. If your data input consists of several values, you can use **Convert from Phys.vi** multiple times and concatenate their outputs.

Convert to Phys.vi

Purpose

Converts a binary representation of a value into its physical value using a type descriptor.

Format



Input

type descriptor is a cluster that specifies the conversion of the binary representation to its physical value:

Start Byte gives the binary representation start byte in the **data in** record.

Byte Length is the binary representation byte length.

Byte Order is the byte ordering of the data in the binary representation:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

Data Type is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths of 4 or 8 are allowed.

Scale Factor defines the physical value scaling:

Phys = (Scale Factor) * (binary representation) + (Scale Offset)

Scale Offset (refer to **Scale Factor**)

data in is the data record from which physical values are to be extracted.

Output

value is the physical value extracted from the record.

Description

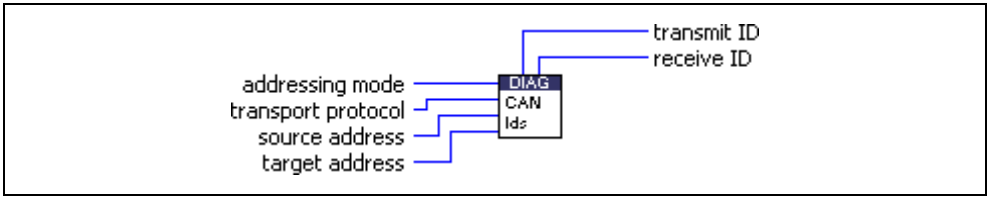
Data output from diagnostic services (for example, **ReadDataByLocalIdentifier.vi**) is usually a byte stream of binary data. If you have a description of the data output (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use **Convert to Phys.vi** to extract the physical value from the byte stream by filling an appropriate type descriptor cluster.

Create Extended CAN IDs.vi

Purpose

Creates diagnostic CAN IDs according to ISO 15765-2.

Format



Input

addressing mode specifies whether the ECU is physically or functionally addressed.

transport protocol specifies whether normal or mixed mode addressing is used.

source address is the logical address of the host (diagnostic tester).

target address is the ECU logical address.

Output

transmit ID is the generated CAN identifier for sending diagnostic request messages from the host to the ECU.

receive ID is the generated CAN identifier for sending diagnostic response messages from the ECU to the host.

Description

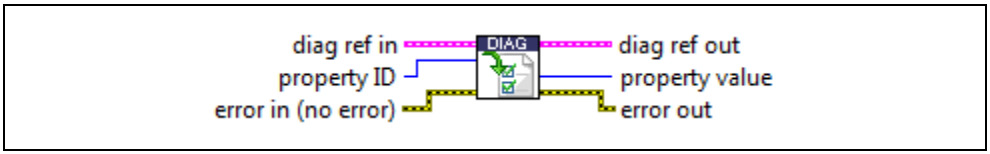
ISO 15765-2 specifies a method (extended/29 bit) of creating CAN identifiers for diagnostic applications given the addressing mode (physical/functional), the transport protocol (normal/mixed), and the 8-bit source and target addresses. This VI implements the construction of these CAN identifiers. You can use them directly in **Open Diagnostic.vi**.

Diag Get Property.vi

Purpose

Gets a diagnostic global internal parameter.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

property ID defines the parameter whose value is to be retrieved. You can create the values using an Enum control.

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms for CAN and 1000 ms for LIN.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms for CAN.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If this value is set to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** returns whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** returns the CAN frame content if filled with defined data or random data bytes.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** returns how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by **success?** = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** is the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

13 **STmin** is the minimum time in seconds between the end of transmission of a frame in a diagnostic request message and the start of transmission of the next frame in the diagnostic request message for LIN-based diagnostic communication.The default is 0.

14 **P2min** is the minimum time in seconds between reception of the last frame of the diagnostic request and the response sent by the node for LIN-based diagnostic communication. The default is 0.05.

15 **Termination** reads the NI-XNET Termination property. Reflections on the CAN and LIN bus can cause communication failures. To prevent reflections, termination can be present as external resistance or resistance the XNET CAN or LIN board applies internally. This property determines whether the XNET board uses termination to the bus. For further information about appropriate terminations of a CAN or LIN network, refer to the *NI-XNET Hardware and Software Manual*. The default is 0.

error in is a cluster that describes error conditions occurring before the VI executes. It is copied unchanged to **error out** and has no other effect on the VI. It is provided for sequencing purposes only.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

property value is the requested property value.

error out describes error conditions. It is copied unchanged from the **error in** cluster. It is provided for sequencing purposes only.

Description

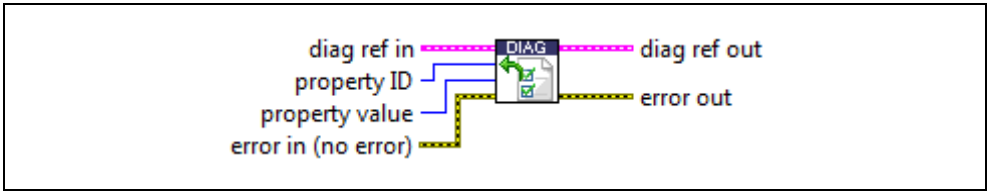
Use this VI to request several internal diagnostic parameters, such as timeouts for the transport protocol. Use **Diag Set Property.vi** to modify them.

Diag Set Property.vi

Purpose

Sets a diagnostic global internal parameter.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

property ID defines the parameter whose value is to be retrieved. You can create the values using an Enum control.

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms for CAN and 1000 ms for LIN.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If this value is set to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** specifies whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** specifies the CAN frame content, filled with defined data or random data.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** specifies how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by **success?** = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** defines the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** sets the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

13 **STmin** sets the minimum time in seconds between the end of transmission of a frame in a diagnostic request message and the start of transmission of the next frame in the diagnostic request message for LIN-based diagnostic communication. The default is 0.

14 **P2min** sets the minimum time in seconds between reception of the last frame of the diagnostic request and the response sent by the node for LIN-based diagnostic communication. The default is 0.05.

15 **Termination** sets the NI-XNET Termination property. Reflections on the CAN and LIN bus can cause communication failures. To prevent reflections, termination can be present as external resistance or resistance the XNET CAN or LIN board applies internally. This property determines whether the XNET board uses termination to the bus. For further information about appropriate terminations of a CAN or LIN network, refer to the *NI-XNET Hardware and Software Manual*. The default is 0.

property value is the value of the property to be set.

error in is a cluster that describes error conditions occurring before the VI executes. It is copied unchanged to **error out** and has no other effect on the VI. It is provided for sequencing purposes only.

Output

diag ref out is a copy of diag ref in. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. It is copied unchanged from the **error in** cluster. It is provided for sequencing purposes only.

Description

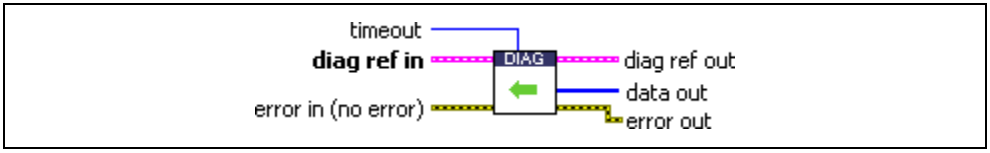
Use this VI to set several internal diagnostic parameters such as timeouts for the transport protocol. Use **Diag Get Property.vi** to read them out.

Diagnostic Frame Recv.vi

Purpose

Receives a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.

Format



Input

timeout specifies the time in milliseconds to wait for a frame on the diagnostic identifier. If no frame arrives within this time, a timeout error is returned.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns up to 8 bytes of payload data from a CAN frame received on the diagnostic identifier.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

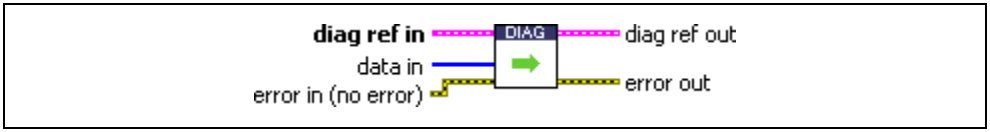
Diagnostic Frame Recv.vi receives an arbitrary raw CAN frame on the diagnostic CAN identifier. For example, you can check the transport protocol implementation of an ECU for correct responses if erroneous protocol requests are issued.

Diagnostic Frame Send.vi

Purpose

Sends a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in is an array of up to 8 bytes sent as a CAN payload on the diagnostic identifier.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

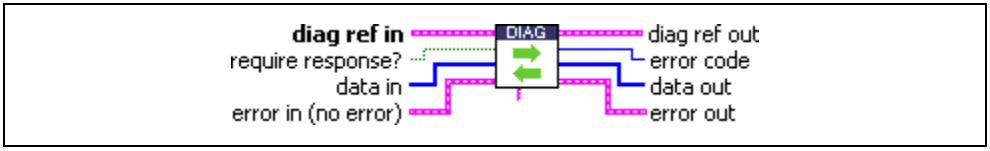
Diagnostic Frame Send.vi transmits an arbitrary raw CAN frame on the diagnostic CAN identifier. For example, you can check the transport protocol implementation of an ECU for robustness if erroneous protocol requests are issued.

Diagnostic Service.vi

Purpose

Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this VI.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

require response? indicates whether a diagnostic service expects a response (TRUE) or not (FALSE). In the latter case, **error code** is returned as 0, and **data out** as an empty array.

data in defines the diagnostic service request message sent to the ECU as a stream of bytes.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error code is the error code sent with a negative response message. In addition, the error cluster indicates an error and gives a more detailed description. If no negative response message occurred, 0 is returned.

data out returns the diagnostic service response message (positive or negative) the ECU sends as a stream of bytes.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Diagnostic Service.vi is a generic routine to execute any diagnostic service. The request and response messages are fed unmodified to the **data in** input and retrieved from the **data out** output, respectively. No interpretation of the contents is done, with one exception: the error number is retrieved from a negative response, if one occurs. In this case, an error also is communicated through the **error out** cluster.

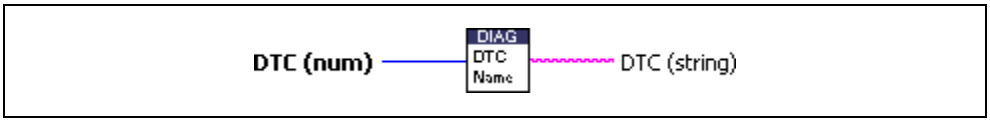
All specialized diagnostic services call **Diagnostic Service.vi** internally.

DTC to String.vi

Purpose

Returns a string representation (such as *P1234*) for a 2-byte Diagnostic Trouble Code (DTC).

Format



Input

DTC (num) is the DTC number as returned in the clusters of **ReadDTCByStatus.vi**, **ReadStatusOfDTC.vi**, **UDS ReportDTCBySeverityMaskRecord.vi**, **UDS ReportDTCByStatusMask.vi**, **UDS ReportSeverityInformationOfDTC.vi**, **UDS ReportSupportedDTCs.vi**, **OBD Request Emission Related DTCs.vi**, or **OBD Request Emission Related DTCs During Current Drive Cycle.vi**.

Note This VI converts only 2-byte DTCs. If you feed in larger numbers, the VI returns garbage.

Output

DTC (string) is the DTC string representation.

Description

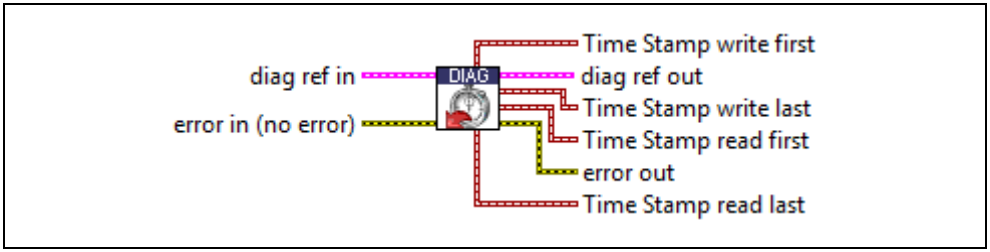
The SAE J2012 standard specifies a naming scheme for 2-byte DTCs consisting of one letter and four digits. Use **DTC to String.vi** to convert a DTC numerical representation to this name.

Get Time Stamp.vi

Purpose

Gets timestamp information about the first/last send/received frame of the ISO TP for CAN and LIN.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on LIN.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Time Stamp write first contains the timestamp of the first write frame. This is usually the FF or SF of the ISO TP.

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

Time Stamp write last contains the timestamp of the last write frame. This is usually the last CF or SF of the ISO TP.

Time Stamp read first contains the timestamp of the first read frame. This is usually the FF or SF of the ISO TP.

Time Stamp read last contains the timestamp of the last read frame. This is usually the CF or SF of the ISO TP.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Get Time Stamp.vi gets the first and last write CAN or LIN frame and the first and last read CAN or LIN frame if the ISO TP transport protocol is used. For all other transport protocols, the timestamps are always 0.

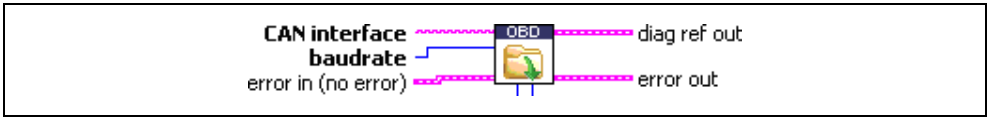
The UDS Read ECU Information example includes an example for getting the timestamp.

OBD Open.vi

Purpose

Opens an OBD-II diagnostic session on a CAN port.

Format



Input

CAN interface specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@nixnet* to the protocol string (for example, *CAN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

Note By selecting *nixnet* as the interface string, the Automotive Diagnostic Command Set uses the Frame Input and Output Queued sessions. To force the use of Frame Input and Output Stream sessions instead, select *ni_genie_nixnet* as the interface string (for example, *CAN1@ni_genie_nixnet*). An application instance can use only one Frame Input Stream Session and one Frame Output Stream Session at a time, so use the default name *nixnet* as the interface string, so that multiple NI-XNET Frame Queued Sessions can coexist on a

single interface, and the Frame Input and Output Stream Sessions may be used, for example, for a Frame logging/replay use case.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

baudrate is the diagnostic communication baud rate.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a cluster containing all necessary diagnostic session information. Wire this cluster as a handle to all subsequent diagnostic VIs and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Use this VI to open a diagnostic communication channel to an ECU for OBD-II. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) in the **diag ref out** cluster, which serves as reference for further diagnostic functions.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware with NI-XNET Frame Input/Output-based sessions:

- **CAN1@nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using NI-XNET hardware with NI-XNET Stream Input/Output-based sessions:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1, c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

First, communication to the ECU is tried on the default 11-bit OBD CAN identifiers; if that fails, the default 29-bit OBD CAN identifiers are tried. If that also fails, the VI returns an error.

You can overwrite the default OBD CAN identifiers optionally with any other identifiers.

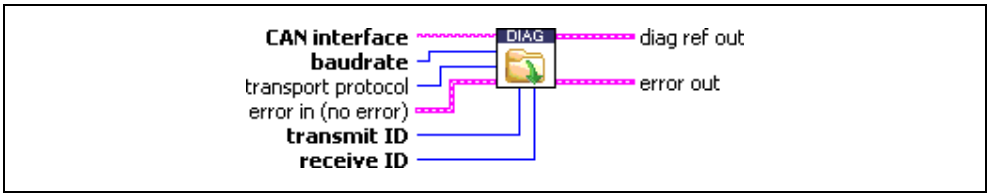
In general, it is not necessary to manipulate the **diag ref out** cluster contents.

Open Diagnostic.vi

Purpose

Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.

Format



Input

CAN interface specifies the CAN interface on which the diagnostic communication should take place. The values are CAN0, CAN1, and so on.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@nixnet* to the protocol string (for example, *CAN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

Note By selecting *nixnet* as the interface string, the Automotive Diagnostic Command Set uses the Frame Input and Output Queued sessions. To force the use of Frame Input and Output Stream sessions instead, select *ni_genie_nixnet* as the interface string (for example, *CAN1@ni_genie_nixnet*). An application instance can use only one Frame Input Stream

Session and one Frame Output Stream Session at a time, so use the default name *nixnet* as the interface string, so that multiple NI-XNET Frame Queued Sessions can coexist on a single interface, and the Frame Input and Output Stream Sessions may be used, for example, for a Frame logging/replay use case.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

baudrate is the diagnostic communication baud rate.

transport protocol specifies the transport protocol for transferring the diagnostic service messages over the CAN network. The following values are valid and can be obtained through an enum control:

0 **ISO TP—Normal Mode**: The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**: The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

2 **VW TP 2.0**

3 **Diagnostic Over IP (DoIP)**: The DoIP TP as specified in ISO 13400 is used.

transmit ID is the CAN identifier for sending diagnostic request messages from the host to the ECU. To specify an extended (29-bit) ID, OR the value with 0x20000000.

receive ID is the CAN identifier or sending diagnostic response messages from the ECU to the host. To specify an extended (29-bit) ID, OR the value with 0x20000000.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a cluster containing all necessary diagnostic session information. Wire this cluster as a handle to all subsequent diagnostic VIs and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Open Diagnostic.vi opens a diagnostic communication channel to an ECU. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) in the **diag ref out** cluster, which serves as reference for further diagnostic functions.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.
- **CAN257**—uses virtual NI-CAN interface 257.

Using NI-XNET hardware with NI-XNET Frame Input/Output-based sessions:

- **CAN1@nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using NI-XNET hardware with NI-XNET Stream Input/Output-based sessions:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1, c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

Note No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call **StartDiagnosticSession.vi** or **UDS DiagnosticSessionControl.vi**.

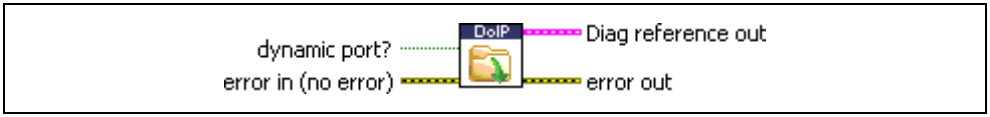
In general, it is not necessary to manipulate the **diag ref out** cluster contents, with one notable exception: If you use the **ISO TP—Mixed Mode** transport protocol, you must store the address extensions for transmit and receive in the appropriate cluster members.

Open Diagnostic on IP.vi

Purpose

Opens a diagnostic session on an IP port. Communication to the ECU is not yet started.

Format



Input

dynamic port defines whether the standard UDP port 13401 (UDP_TEST_EQUIPMENT_LISTEN) is used for communication (FALSE) or a dynamically assigned UDP port (UDP_TEST_EQUIPMENT_REQUEST) is opened (TRUE). Default is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a cluster containing all necessary information about the diagnostic session. Wire this output as a handle to all subsequent diagnostic VIs, and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Open Diagnostic on IP.vi opens a Diagnostic on Internet Protocol (DoIP) communication channel to an ECU. The UDP port specified as input is initialized, and a handle to it is stored (among other internal data) in the **Diag reference out** cluster, which serves as reference for further diagnostic functions.

Note that no communication to an ECU takes place at this point. To open a diagnostic session on an ECU, call **DoIP Get Entities.vi** to find out which DoIP entities (DoIP-capable ECUs) exist in the network. You need to create a TCP/IP connection to the selected DoIP entity using **DoIP Connect.vi**. After that, you can execute diagnostic services on the TCP/IP connection.

This VI replaces the standard (CAN-based) **Open Diagnostic.vi**, because the CAN parameters are no longer relevant for IP-based diagnostics.

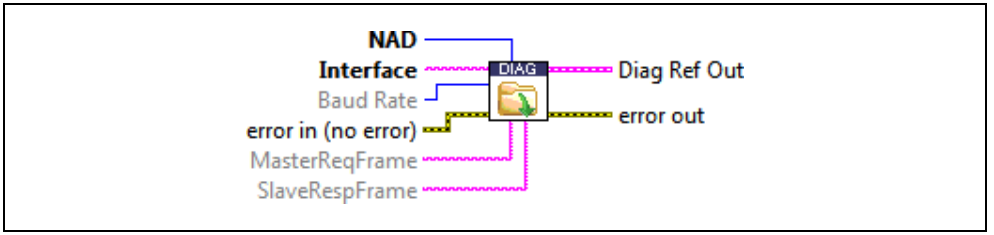
In general, you do not need to manipulate the **Diag reference out** cluster contents.

Open Diagnostic on LIN.vi

Purpose

Opens a diagnostic session on an NI-XNET LIN port. Communication to the ECU is not yet started.

Format



Input

Interface specifies the LIN interface on which the diagnostic communication should take place, and points to the corresponding database cluster. The values for the XNET hardware interface names are LIN1, LIN2, and so on.

The Automotive Diagnostic Command Set supports NI-XNET LIN devices for LIN communication only. To use your NI-XNET interface, define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@nixnet* to the protocol string (for example, *LIN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

The Automotive Diagnostic Command Set requires valid assignments to a LIN database such as LDF or FIBEX. To communicate with hardware products on the external network, applications must understand how that hardware communicates in the actual embedded system, such as the vehicle. This embedded communication is described within a standardized file, such as FIBEX (`.xml`) or LDF (`.ldf`) for LIN. Within NI-XNET, this file is referred to as a database. The database contains many object classes, each of which describes a distinct entity in the embedded system.

For LIN, select a LIN database and cluster to assign all settings from the selected cluster automatically, such as the LIN Baudrate, Master Request Frame, Slave Response Frame, or LIN Diagnostic Schedule.

Using NI-XNET hardware, the Interface string should look like the following examples:

- **LIN1@nixnet:XNET_LIN_Database**—Uses LIN interface 1 of an NI-XNET device and assigns properties such as baudrate automatically from the XNET alias XNET_LIN_Database.
- **LIN2@nixnet:XNET_LIN_Database**—Uses LIN interface 2 of an NI-XNET device and so on, with the form *LINx*.

Refer to the *NI-XNET Hardware and Software Manual* to assign a database cluster alias.

MasterReqFrame selects the Master Request Frame from an LDF or FIBEX database. If you assign an empty string (default) as **MasterReqFrame**, the name as defined in the LIN MasterReq standard is used.

SlaveRespFrame selects the Slave Response Frame from an LDF or FIBEX database. If you assign an empty string (default) as **SlaveRespFrame**, the name as defined in the LIN SlaveResp standard is used.

Baud Rate is the diagnostic communication baud rate. The default is –1, which reuses the baudrate of the selected LIN cluster from the assigned FIBEX or LDF database. To change the baudrate from the database, select a valid LIN baudrate.

NAD is the address of the slave node being addressed in a request. NAD also indicates the source of a response. NAD values are 1–127 (0x7F), while 0 (zero) and 128 (0x80)–255 (0xFF) are reserved for other purposes.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag Ref Out is a cluster containing all necessary information about the diagnostic session. Wire this output as a handle to all subsequent diagnostic VIs, and close it using **Close Diagnostic.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Open Diagnostic on LIN.vi opens a diagnostic LIN communication channel to an ECU. The LIN port specified as input is initialized, and a handle to it is stored (among other internal data) in the **diag ref out** cluster, which serves as reference for further diagnostic functions.

A possible example of selections for the interface parameter for the NI-XNET hardware targets is:

- **LIN1@nixnet:[LIN Cluster]**—Uses LIN interface 1 of an NI-XNET device and settings of the LIN database of [LIN Cluster] alias as defined by NI-XNET.

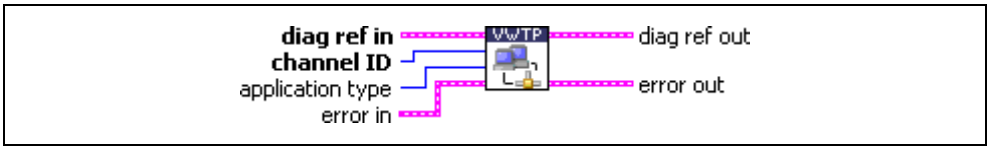
Note No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call **StartDiagnosticSession.vi** or **UDS DiagnosticSessionControl.vi**. **Open Diagnostic on LIN.vi** supports only NI-XNET LIN hardware. A valid LIN cluster of a LDF or FIBEX database must be assigned also.

VWTP Connect.vi

Purpose

Establishes a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

channel ID defines the CAN identifier on which the ECU responds for this connection. The ECU defines the ID on which the host transmits.

application type specifies the type of communication that takes place on the communication channel. For diagnostic applications, specify *KWP2000 (1)*. The other values are for manufacturer-specific purposes.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must establish a connection to the ECU before any diagnostic communication can occur. This VI sets up a unique communication channel to an ECU for subsequent diagnostic service requests.

Note You must maintain the communication link you created by periodically (at least once a second) calling **VWTP Connection Test.vi**.

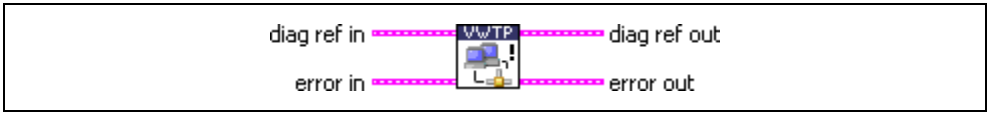
There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

VWTP Connection Test.vi

Purpose

Maintains a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must periodically maintain the connection link to the ECU so that the ECU does not terminate it.

This VI sends a Connection Test message to the ECU and evaluates its response, performing the steps necessary to maintain the connection.

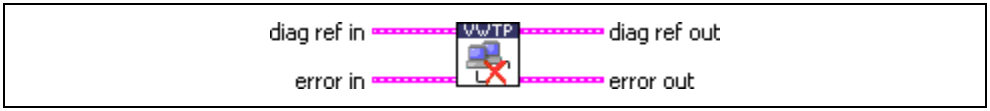
There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

VWTP Disconnect.vi

Purpose

Terminates a connection channel to an ECU using the VW TP 2.0.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

For the VW TP 2.0, you must disconnect the connection link to the ECU to properly terminate communication to the ECU. This VI sends the proper disconnect messages and unlinks the communication.

You can create a new connection to the same ECU using **VWTP Connect.vi** again.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

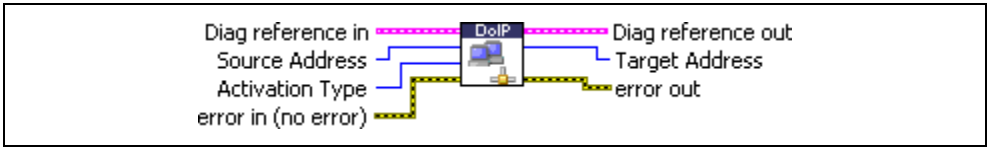
DoIP Functions

DoIP Activate Routing.vi

Purpose

Defines the source and target addresses for a DoIP TCP/IP connection.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

Source Address is the DoIP source address of the tester that starts the communication.

Activation Type indicates the specific type of routing activation that may require different types of authentication and/or confirmation. Defined values are:

0 Default.

1 WWH-OBD (worldwide harmonized onboard diagnostic).

0xE0 Use an OEM-pecific central security approach.

Values 2 to 0xDF are reserved. Values 0xE0 to 0xFF are OEM specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning:

the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

Target Address is the logical address of the responding DoIP entity.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

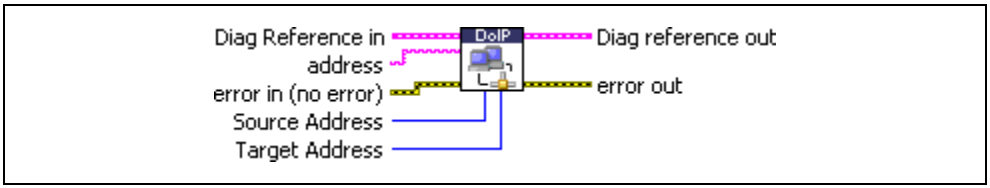
DoIP Activate Routing.vi establishes a route for the DoIP messages and assigns an endpoint **Target Address**. After successfully establishing a route, diagnostic messages can be exchanged with the target DoIP entity using any diagnostic service VI.

DoIP Connect.vi

Purpose

Creates a TCP/IP connection to a DoIP entity identified by its IP address.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address is the IP address of the DoIP entity to connect to (a string in *a.b.c.d* notation).

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Source Address is the DoIP source address of the tester that starts the communication. You can leave this input unwired if you are activating a route through **DoIP Activate Routing.vi**.

Target Address is the DoIP target address of the device under test that should be connected to. You can leave this input unwired if you are activating a route through **DoIP Activate Routing.vi**.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

DoIP Connect.vi creates a unique TCP/IP data connection to a certain DoIP entity identified by its IP **address**. The IP address might be retrieved from **DoIP Get Entities.vi**. The TCP/IP data connection is needed to exchange diagnostic service requests.

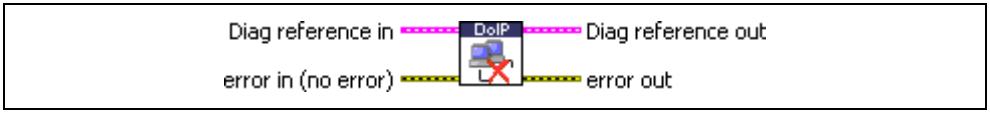
You can specify **Source Address** and **Target Address** at this point or leave them blank if a routing activation is executed later using **DoIP Activate Routing.vi**.

DoIP Disconnect.vi

Purpose

Disconnects the TCP/IP connection to a DoIP entity.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

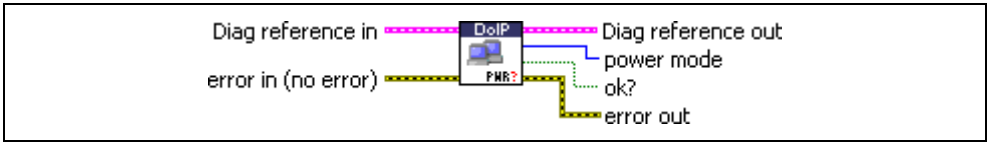
DoIP Disconnect.vi terminates the TCP/IP connection to the connected DoIP entity. After executing this VI, diagnostic services no longer can be executed on that DoIP entity. You can reconnect with **DoIP Connect.vi**.

DoIP Get Diagnostic Power Mode.vi

Purpose

Gets information about the DoIP entity power state.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

power mode identifies whether the vehicle is in Diagnostic Power Mode and ready to perform reliable diagnostics. Possible values are:

0 Not ready

1 Ready

All other values are reserved.

ok? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

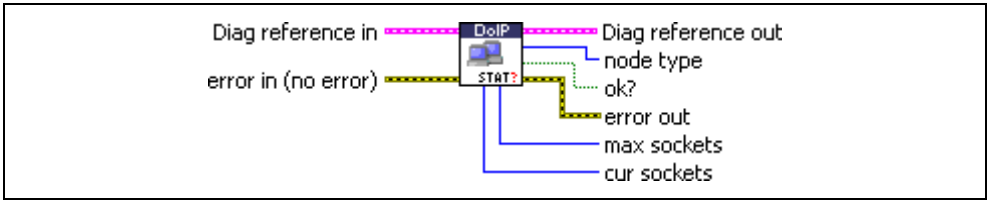
DoIP Get Diagnostic Power Mode.vi retrieves the Diagnostic Power Mode of a vehicle. For example, test equipment can use this information to verify whether the vehicle is in Diagnostic Power Mode, which allows for performing reliable diagnostics on the vehicle's components.

DoIP Get DoIP Entity Status.vi

Purpose

Gets status information from a DoIP entity.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

node type is a U8 ring that indicates the type of DoIP entity. Possible values are:

0 DoIP gateway

1 DoIP node

All other values are reserved.

ok? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

max sockets represents the maximum number of concurrent TCP/IP sockets allowed with this DoIP entity excluding the reserve socket required for socket handling.

cur sockets is the number of currently established TCP/IP sockets.

Description

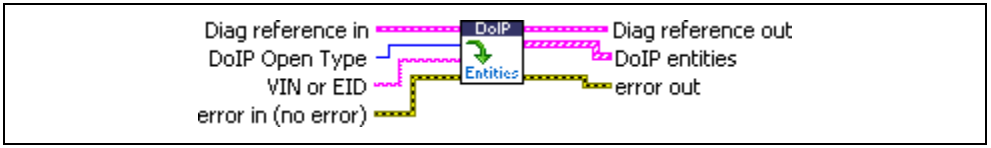
DoIP Get DoIP Entity Status.vi identifies certain operating conditions of the responding DoIP entity. For example, this allows for test equipment to detect existing diagnostic communication sessions as well as the capabilities of a DoIP entity.

DoIP Get Entities.vi

Purpose

Returns a table of all DoIP entities (vehicles) on the local subnet, possibly restricted to EID or VIN.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DoIP Open Type is a U16 ring. It defines which DoIP entities this command queries and lists. Allowed values are *VIN*, *EID*, and *All*.

VIN or EID depends on the DoIP Open Type:

DoIP Open Type	VIN or EID Value
VIN	**VIN or EID** is a 17-character Vehicle Identification Number. Only DoIP entities for this VIN are listed.
EID	**VIN or EID** is an Entity ID (usually a MAC address). Only the DoIP entity with this ID is listed. Specify the EID as *xx-xx-xx-xx-xx-xx,* where each *x* is a hexadecimal digit.
All	**VIN or EID** is ignored.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not

execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

DoIP Entities is an array of clusters, each of which contains the description of one DoIP entity that responded to the command:

VIN is the 17-character Vehicle Identification Number of the DoIP entity. It can be blank if the DoIP entity does not yet belong to a vehicle.

Source Address is the 16-bit DoIP address of this entity. This address can distinguish multiple DoIP entities within a vehicle.

EID is a 6-byte array of the Entity ID, which is usually the DoIP device MAC address.

GID is a unique 6-byte group identification of DoIP entities that belong to the same vehicle. It is used as long as a VIN is not yet defined.

IP Address is the IP Address of this DoIP entity in a.b.c.d notation. Use this IP address to connect to the DoIP entity using **DoIP Connect.vi**.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

DoIP Get Entities.vi uses a UDP broadcast to identify all DoIP entities in the local subnet matching a certain condition. The entities responding are returned in the **DoIP Entities** cluster array.

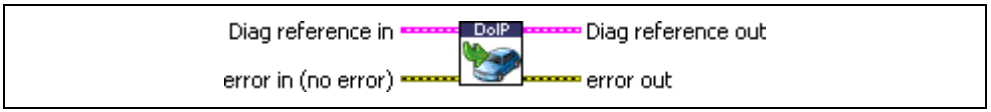
The conditions are either a common VIN or EID or simply all entities connected. Refer to the **DoIP Open Type** and **VIN or EID** descriptions.

DoIP Send Vehicle Identification Request.vi

Purpose

Sends a UDP request to all DoIP-capable vehicles in the local subnet to identify themselves.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

DoIP Send Vehicle Identification Request.vi sends a Vehicle Identification Request to all DoIP entities in the local subnet.

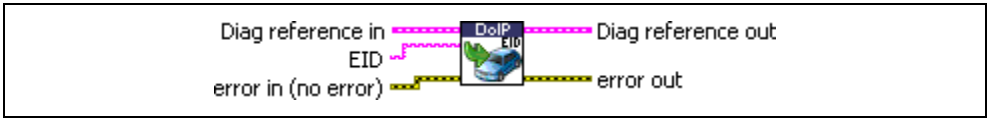
Usually, this is done as part of **DoIP Get Entities.vi** and does not need to be executed separately.

DoIP Send Vehicle Identification Request w EID.vi

Purpose

Sends a UDP request to all DoIP-capable vehicles with a certain EID (MAC address) in the local subnet to identify themselves.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

EID is the Entity ID (usually the MAC address) of the DoIP entity assumed to respond. Specify the EID as *xx-xx-xx-xx-xx-xx*, where each *x* stands for a hexadecimal digit.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

DoIP Send Vehicle Identification Request w EID.vi sends a Vehicle Identification Request to all DoIP entities in the local subnet identified by the given **EID**.

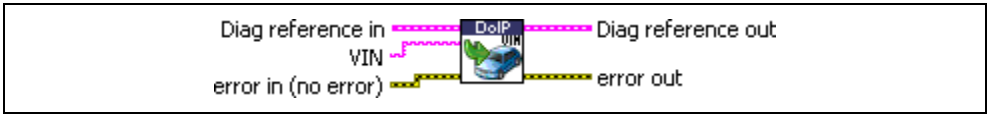
Usually, this is done as part of **DoIP Get Entities.vi** and does not need to be executed separately.

DoIP Send Vehicle Identification Request w VIN.vi

Purpose

Sends a UDP request to all DoIP-capable vehicles with a certain VIN (Vehicle Identification Number) in the local subnet to identify themselves.

Format



Input

Diag reference in specifies the diagnostic session handle, obtained from **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

VIN is the 17-character Vehicle Identification Number of the DoIP entity assumed to respond.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Diag reference out is a copy of **Diag reference in**. You can wire it to subsequent diagnostic VIs.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

DoIP Send Vehicle Identification Request w VIN.vi sends a Vehicle Identification Request to all DoIP entities in the local subnet identified by the given **VIN**.

Usually, this is done as part of **DoIP Get Entities.vi** and does not need to be executed separately.

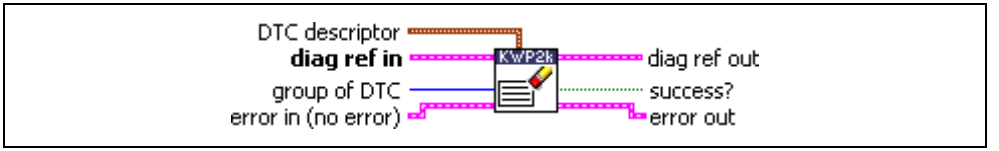
KWP2000 Services

ClearDiagnosticInformation.vi

Purpose

Executes the ClearDiagnosticInformation service and clears selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be cleared. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI clears the diagnostic information on the ECU memory. If the **group of DTC** parameter is present, the ECU is requested to clear all memory including the DTCs.

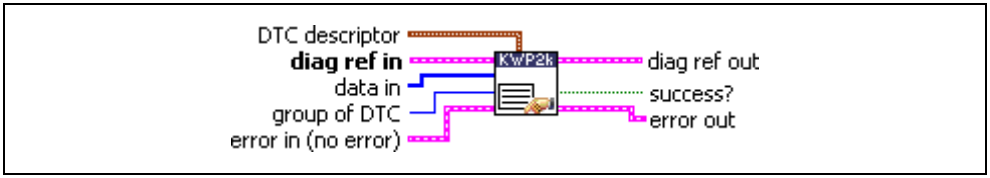
For further details about this service, refer to the ISO 14230-3 standard.

ControlDTCSetting.vi

Purpose

Executes the ControlDTCSetting service and modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in specifies application-specific data that control DTC generation.

group of DTC specifies the group of Diagnostic Trouble Codes to be controlled. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

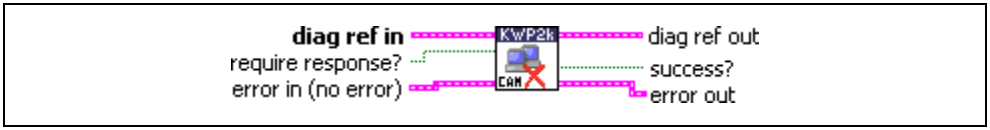
source identifies the VI where the error occurred.

DisableNormalMessageTransmission.vi

Purpose

Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

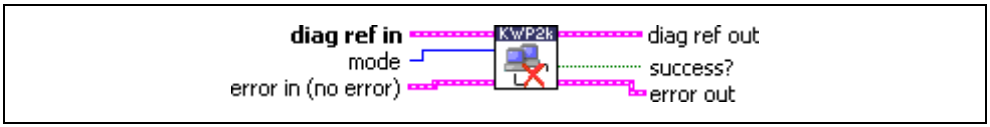
source identifies the VI where the error occurred.

ECUReset.vi

Purpose

Executes the ECUReset service. Resets the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the reset mode:

Hex	Description
01	**PowerOn** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the client (tester) re-establishes communication.
02	**PowerOnWhileMaintainingCommunication** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the server (ECU) maintains communication with the client (tester).
03–7F	**Reserved**
80–FF	**ManufacturerSpecific** This range of values is reserved for vehicle manufacturer-specific use.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

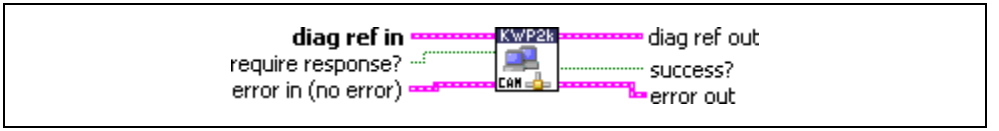
This VI requests the ECU to perform an ECU reset effectively based on the **mode** parameter value content. The vehicle manufacturer determines when the positive response message is sent.

EnableNormalMessageTransmission.vi

Purpose

Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

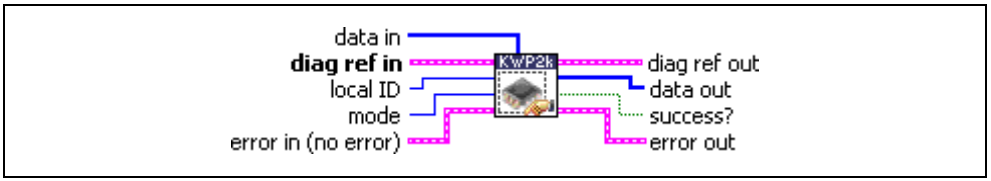
source identifies the VI where the error occurred.

InputOutputControlByLocalIdentifier.vi

Purpose

Executes the InputOutputControlByLocalIdentifier service. Modifies ECU I/O port behavior.

Format



Input

data in defines application-specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the I/O to be manipulated. The values are application specific.

mode defines the type of I/O control. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ReportCurrentState

4: ResetToDefault

5: FreezeCurrentState

7: ShortTermAdjustment

8: LongTermAdjustment

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the **local ID** parameter.

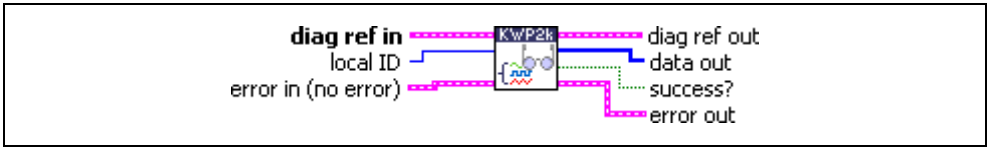
For further details about this service, refer to the ISO 14230-3 standard.

ReadDataByLocalIdentifier.vi

Purpose

Executes the ReadDataByLocalIdentifier service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the data to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the data record from the ECU. If you know the record data description, you can interpret this record using **Convert from Phys.vi**.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests data record values from the ECU identified by the **local ID** parameter.

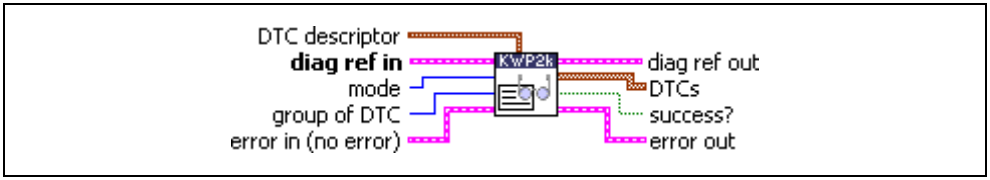
For further details about this service, refer to the ISO 14230-3 standard.

ReadDTCByStatus.vi

Purpose

Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode defines the type of DTCs to be read. The values are application specific. The usual values are:

2: AllIdentified

3: AllSupported

group of DTC specifies the group of Diagnostic Trouble Codes to be read. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**)

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI reads DTCs by status from the ECU memory. If you use the optional **group of DTC** parameter, the ECU reports DTCs only with status information based on the functional group selected by **group of DTC**.

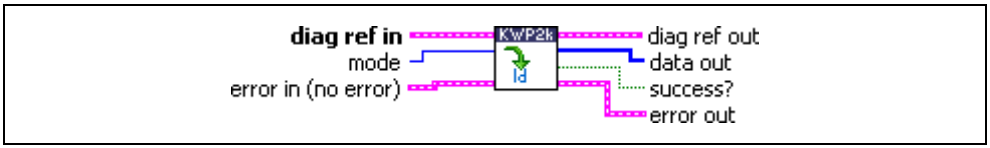
For further details about this service, refer to the ISO 14230-3 standard.

ReadECUIdentification.vi

Purpose

Executes the ReadECUIdentification service. Returns ECU identification data.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the type of identification information to be returned. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU identification data.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests identification data from the ECU. The **mode** parameter identifies the type of identification data requested. The ECU returns identification data that the **data out** parameter can access. The **data out** format and definition are vehicle manufacturer specific.

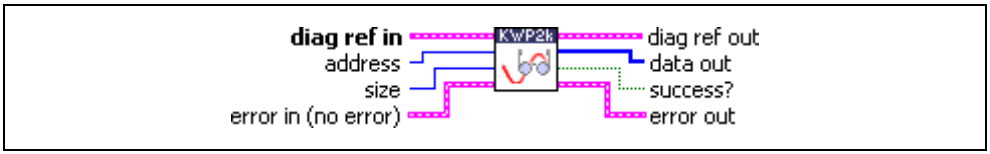
For further details about this service, refer to the ISO 14230-3 standard.

ReadMemoryByAddress.vi

Purpose

Executes the ReadMemoryByAddress service. Reads data from the ECU memory.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address from which data are to be read. Notice that only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

Data out returns the memory data from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests memory data from the ECU identified by the **address** and **size** parameters. The **data out** format and definition are vehicle manufacturer specific. **data out** includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

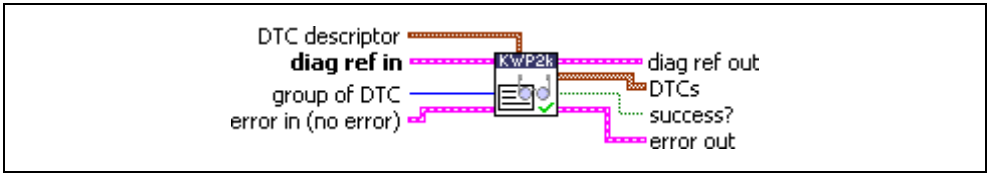
For further details about this service, refer to the ISO 14230-3 standard.

ReadStatusOfDTC.vi

Purpose

Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be read. The following values have a special meaning, and you can specify them through a ring control:

0x0000 All powertrain DTCs

0x4000 All chassis DTCs

0x8000 All body DTCs

0xC000 All network-related DTCs

0xFF00 All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle

2 pendingDTC

3 confirmedDTC

4 testNotCompletedSinceLastClear

5 testFailedSinceLastClear

6 testNotCompletedThisMonitoringCycle

7 warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**)

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI reads diagnostic trouble codes from the ECU memory. If you use the optional **group of DTC** parameter, the ECU reports DTCs based only on the functional group selected by **group of DTC**.

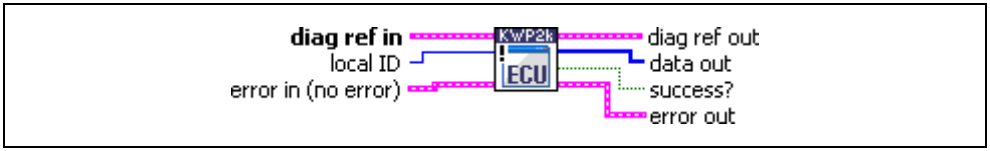
For further details about this service, refer to the ISO 14230-3 standard.

RequestRoutineResultsByLocalIdentifier.vi

Purpose

Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine from which this VI retrieves results. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests results (for example, exit status information) referenced by **local ID** and generated by the routine executed in the ECU memory.

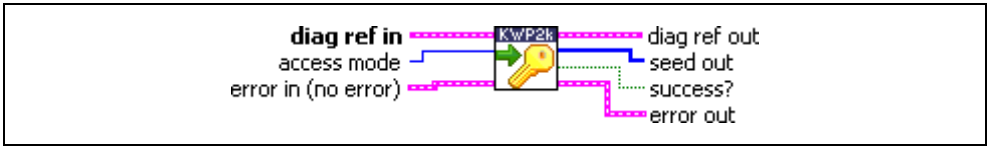
For further details about this service, refer to the ISO 14230-3 standard.

RequestSeed.vi

Purpose

Executes the SecurityAccess service to retrieve a seed from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

seed out returns the seed from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

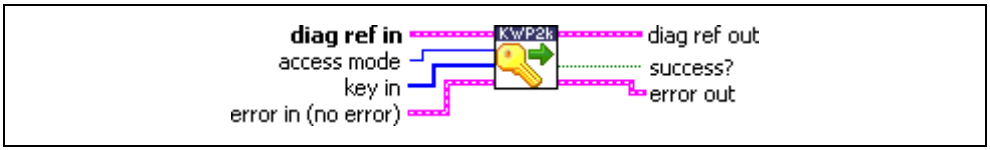
1. Request a seed from the ECU using **RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using SendKey.vi with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

SendKey.vi

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

key in defines the key data to be sent to the ECU.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

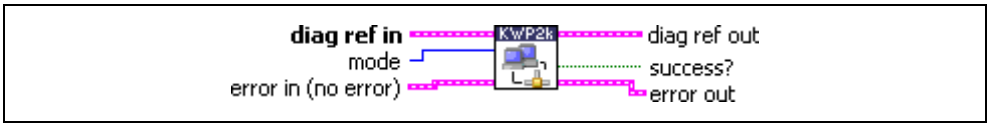
1. Request a seed from the ECU using **RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

StartDiagnosticSession.vi

Purpose

Executes the StartDiagnosticSession service. Sets up the ECU in a specific diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the diagnostic mode into which the ECU is brought. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI enables different diagnostic modes in the ECU. The possible diagnostic modes are not defined in the ISO 14230 standard and are application specific. A diagnostic session starts only if communication with the ECU is established. For more details about starting communication, refer to the ISO 14230-2 standard. If no diagnostic session has been requested after **Open Diagnostic.vi**, a default session is automatically enabled in the ECU. The default session supports at least the following services:

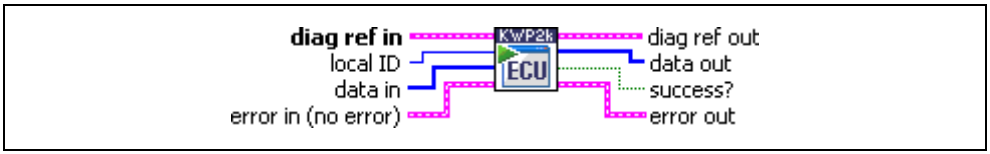
- The StopCommunication service (refer to **Close Diagnostic.vi** and the ISO 14230-2 standard).
- The TesterPresent service (refer to **TesterPresent.vi** and the ISO 14230-3 standard).

StartRoutineByLocalIdentifier.vi

Purpose

Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine to be started. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI starts a routine in the ECU memory. The routine in the ECU starts after the positive response message is sent. The routine stops until **StopRoutineByLocalIdentifier.vi** is issued. The routines could be either tests run instead of normal operating code or routines enabled and executed with the normal operating code running. In the first case, you may need to switch the ECU to a specific diagnostic mode using **StartDiagnosticSession.vi** or unlock the ECU using the SecurityAccess service prior to using **StartRoutineByLocalIdentifier.vi**.

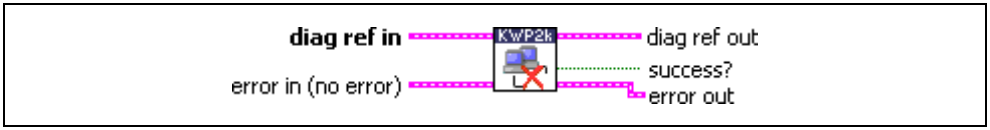
For further details about this service, refer to the ISO 14230-3 standard.

StopDiagnosticSession.vi

Purpose

Executes the StopDiagnosticSession service. Returns the ECU to normal mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

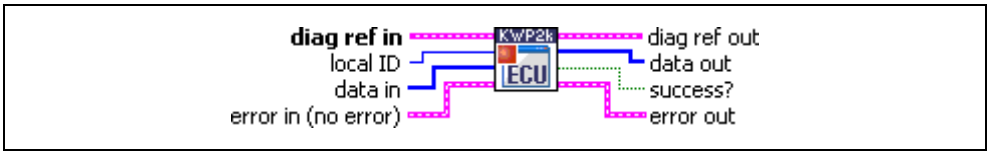
This VI disables the current ECU diagnostic mode. A diagnostic session stops only if communication is established with the ECU and a diagnostic session is running. If no diagnostic session is running, the default session is active. **StopDiagnosticSession.vi** cannot disable the default session. If the ECU has stopped the current diagnostic session, it performs the necessary action to restore its normal operating conditions. Restoring the normal operating conditions of the ECU may include resetting all controlled actuators if they were activated during the diagnostic session being stopped, and resuming all normal ECU algorithms. You should call **StopDiagnosticSession.vi** before disabling communication with **Close Diagnostic.vi**, but only if you previously used **StartDiagnosticSession.vi**.

StopRoutineByLocalIdentifier.vi

Purpose

Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the routine to be stopped. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI stops a routine in the ECU memory referenced by the **local ID** parameter.

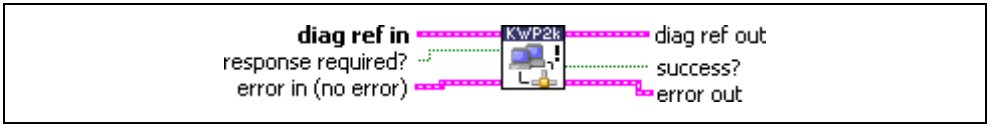
For further details about this service, refer to the ISO 14230-3 standard.

TesterPresent.vi

Purpose

Executes the TesterPresent service. Keeps the ECU in diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

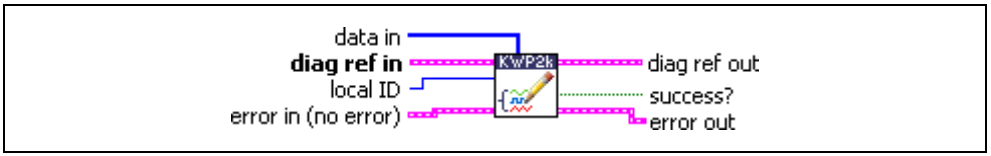
Keep calling **TesterPresent.vi** within the ECU timeout period if no other service is executed.

WriteDataByLocalIdentifier.vi

Purpose

Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.

Format



Input

data in defines the data record written to the ECU. If you know the record data description, you can use **Convert from Phys.vi** to generate this record.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

local ID defines the local identifier of the data to be written. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the KWP2000 WriteDataByLocalIdentifier service and writes RecordValues (data values) to the ECU. **data in** identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

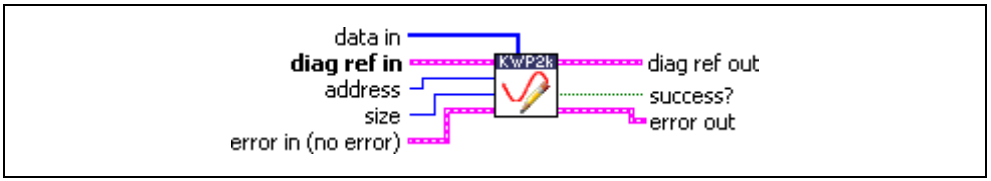
For further details about this service, refer to the ISO 14230-3 standard.

WriteMemoryByAddress.vi

Purpose

Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

Format



Input

data in defines the memory block to be written to the ECU.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address to which data are written. Notice that only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the KWP2000 WriteDataByAddress service and writes RecordValues (data values) to the ECU. **address** and **size** identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

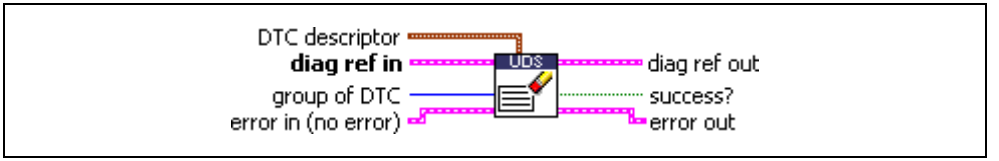
UDS (DiagOnCAN) Services

UDS ClearDiagnosticInformation.vi

Purpose

Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter basically to convert the **group of DTC** parameter to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

group of DTC specifies the group of Diagnostic Trouble Codes to be cleared. The values are application specific. The following value has a special meaning, and you can specify it through a ring control:

0xFFFFFF All DTCs

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI clears the diagnostic information on the ECU memory. If the **group of DTC** parameter is present, the ECU is requested to clear all memory including the DTCs.

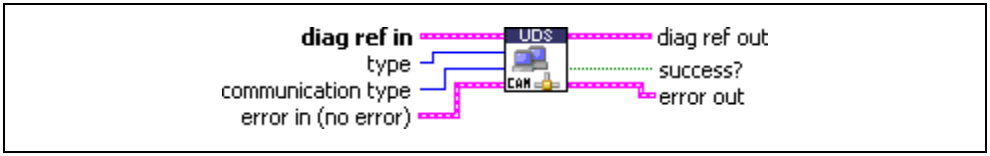
For further details about this service, refer to the ISO 15765-3 standard.

UDS CommunicationControl.vi

Purpose

Executes the UDS CommunicationControl service. Use this VI to switch transmission and/or reception of the normal communication messages (usually CAN messages) on or off.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

type indicates whether transmission/reception is to be switched on/off. The usual values are:

00: enableRxAndTx

01: enableRxAndDisableTx

02: disableRxAndEnableTx

03: disableRxAndTx

communication type is a bitfield indicating the application level to change. The usual values are:

01: application

02: networkManagement

You can change more than one level at a time.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

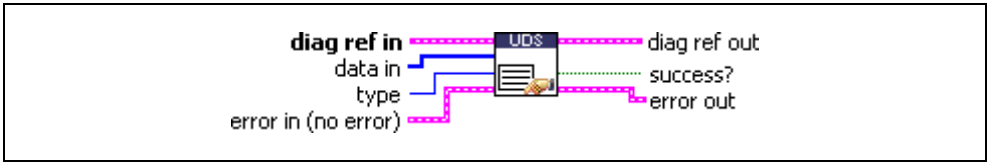
This VI executes the UDS CommunicationControl service and switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off. The **type** and **communication type** parameters are vehicle manufacturer specific (one OEM may disable the transmission only, while another OEM may disable the transmission and the reception based on vehicle manufacturer specific needs). The request is either transmitted functionally addressed to all ECUs with a single request message, or transmitted physically addressed to each ECU in a separate request message.

UDS ControlDTCSetting.vi

Purpose

Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) generation behavior.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in specifies application-specific data that control DTC generation.

type specifies the control mode:

1: on

2: off

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

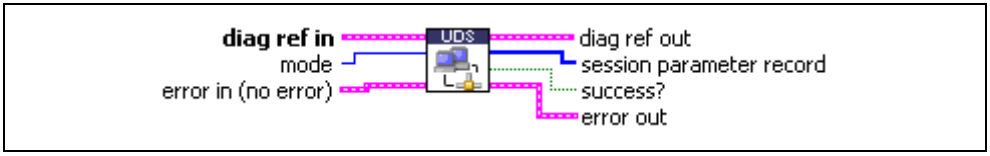
source identifies the VI where the error occurred.

UDS DiagnosticSessionControl.vi

Purpose

Executes the UDS DiagnosticSessionControl service. Sets up the ECU in a specific diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the diagnostic mode into which the ECU is brought. The values are application specific. The usual values are:

01: defaultSession

02: ECUProgrammingSession

03: ECUExtendedDiagnosticSession

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

session parameter record returns implementation-dependent data from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

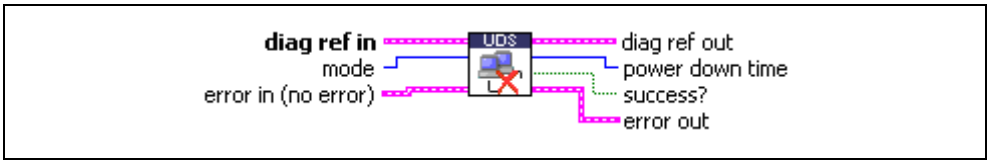
source identifies the VI where the error occurred.

UDS ECUReset.vi

Purpose

Executes the UDS ECUReset service. Resets the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode indicates the reset mode:

Hex	Description
01	hardReset
02	keyOffOnReset
03	softReset
04	enableRapidPowerShutDown
05	disableRapidPowerShutDown

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to

a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

power down time returns the minimum standby sequence time that the server remains in the power-down sequence in seconds. A value of FF hex indicates a failure or time not available.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests the ECU to perform an ECU reset effectively based on the **mode** parameter value content. The vehicle manufacturer determines when the positive response message is sent.

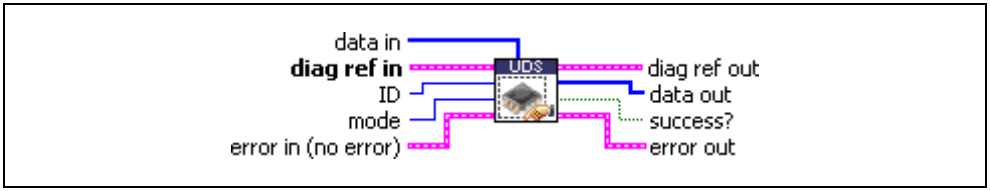
For further details about this service, refer to the ISO 15765-3 standard.

UDS InputOutputControlByIdentifier.vi

Purpose

Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.

Format



Input

data in defines application specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the I/O to be manipulated. The values are application specific.

mode defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ResetToDefault

2: FreezeCurrentState

3: ShortTermAdjustment

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the **local ID** parameter.

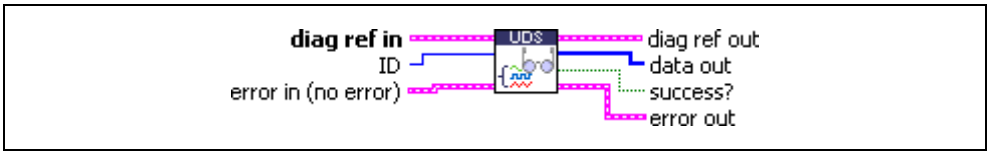
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReadDataByIdentifier.vi

Purpose

Executes the UDS ReadDataByIdentifier service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the data to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the data record from the ECU. If you know the record data description, you can use **Convert to Phys.vi** to interpret this record.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests data record values from the ECU identified by the **ID** parameter.

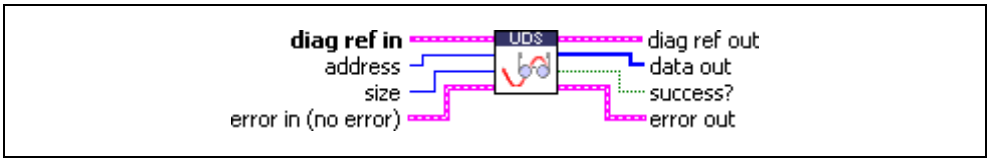
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReadMemoryByAddress.vi

Purpose

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address from which data are to be read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU memory data.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI requests ECU memory data identified by the **address** and **size** parameters. The **data out** format and definition are vehicle manufacturer specific. **data out** includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

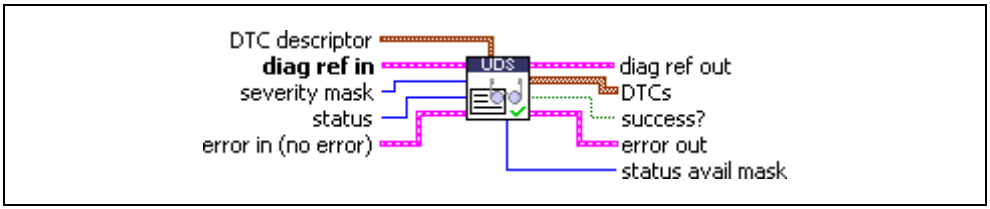
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportDTCBySeverityMaskRecord.vi

Purpose

Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

severity mask defines the status of DTCs to be read. The values are application specific.

status defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear

6 testNotCompletedThisMonitoringCycle

7 warningIndicatorRequested

Add Data contains optional additional data for this DTC.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs.

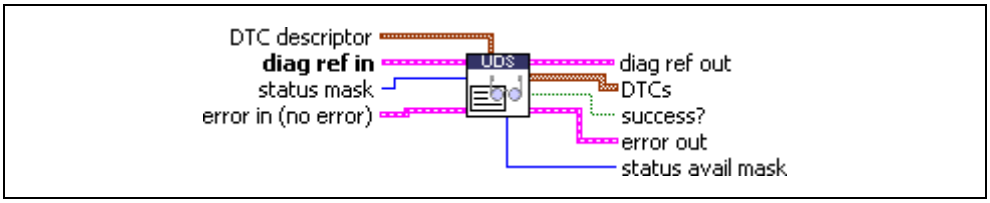
For further details about this service, refer to the ISO 14229-1 standard.

UDS ReportDTCByStatusMask.vi

Purpose

Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

status mask defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU.

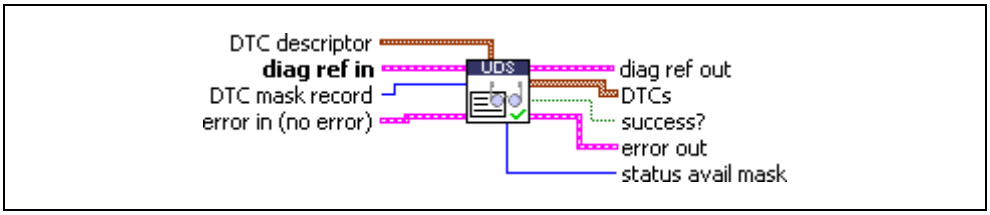
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportSeverityInformationOfDTC.vi

Purpose

Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DTC mask record defines the status of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU memory.

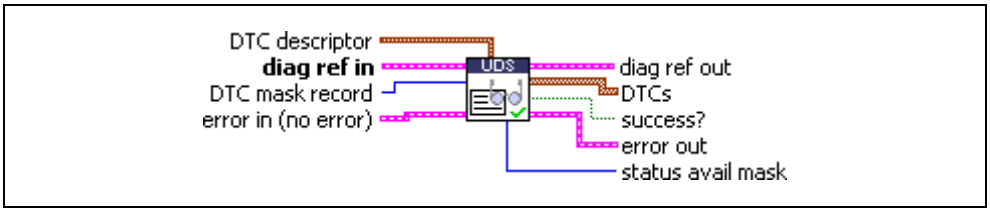
For further details about this service, refer to the ISO 15765-3 standard.

UDS ReportSupportedDTCs.vi

Purpose

Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

status avail mask is an application-specific value returned for all DTCs.

Description

This VI executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads all supported DTCs from the ECU memory.

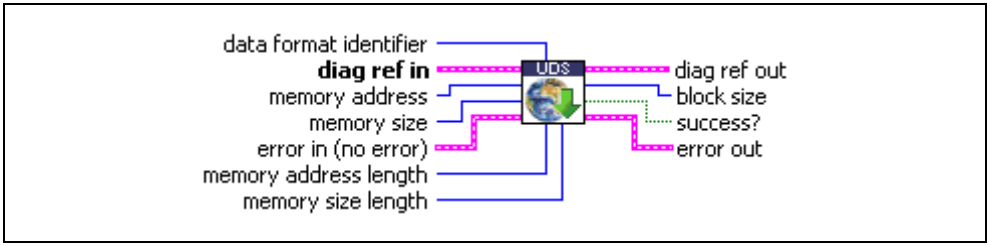
For further details about this service, refer to the ISO 15765-3 standard.

UDS RequestDownload.vi

Purpose

Initiates a download of data to the ECU.

Format



Input

data format identifier defines the compression and encryption scheme to be used for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

diag ref in specifies the handle for the diagnostic session. This is obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address to which data are written.

memory size defines the size of the data to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length defines the number of bytes of the **memory address** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

memory size length defines the number of bytes of the **memory size** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

Output

diag ref out is a copy of **diag ref in.** It can be wired to subsequent diagnostic VIs.

block size returns the number of data bytes to be transferred to the ECU in subsequent **UDS TransferData.vi** requests.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

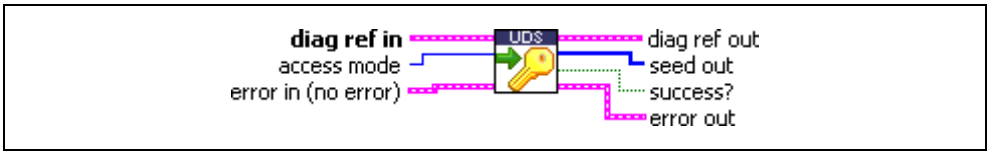
UDS RequestDownload.vi initiates the download of a data block to the ECU. This is required to set up the download process; the actual data transfer occurs with subsequent **UDS TransferData.vi** requests. The transfer must occur in blocks of the size that this service returns (the **block size** parameter). After the download completes, use the **UDS RequestTransferExit.vi** service to terminate the process.

UDS RequestSeed.vi

Purpose

Executes the UDS SecurityAccess service to retrieve a seed from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

seed out returns the seed from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

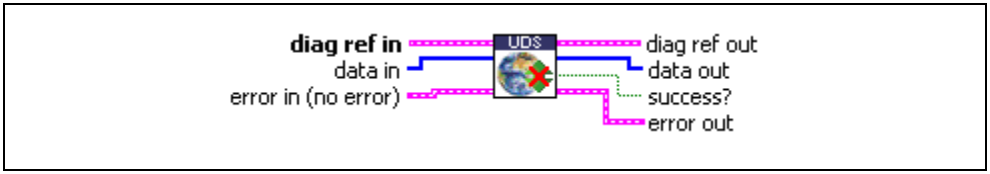
1. Request a seed from the ECU using **UDS RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **UDS SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

UDS RequestTransferExit.vi

Purpose

Terminates a download/upload process.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in defines a data record to be written to the ECU as part of the termination process. The meaning is implementation dependent; this might be a checksum or a similar verification instrument.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. It can be wired to subsequent diagnostic VIs.

data out returns a memory data block from the ECU as part of the termination process. The meaning is implementation dependent; this might be a checksum or a similar verification instrument.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

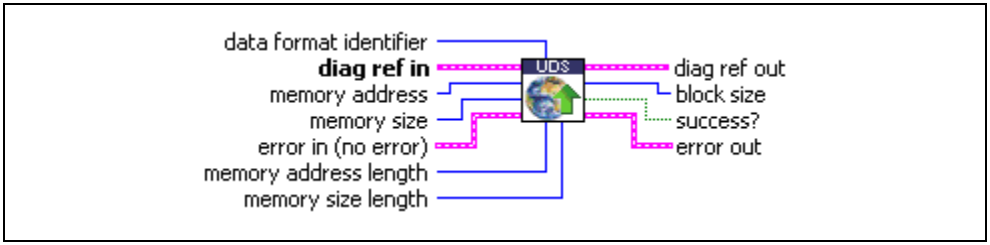
UDS RequestTransferExit.vi terminates a download or upload process initialized with **UDS RequestDownload.vi** or **UDS RequestUpload.vi**.

UDS RequestUpload.vi

Purpose

Initiates an upload of data from the ECU.

Format



Input

data format identifier defines the compression and encryption scheme to be used for the data blocks read from the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

diag ref in specifies the handle for the diagnostic session. This is obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address from which data are read.

memory size defines the size of the data to be read.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length defines the number of bytes of the **memory address** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

memory size length defines the number of bytes of the **memory size** parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

block size returns the number of data bytes to be transferred from the ECU in subsequent **UDS TransferData.vi** requests.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

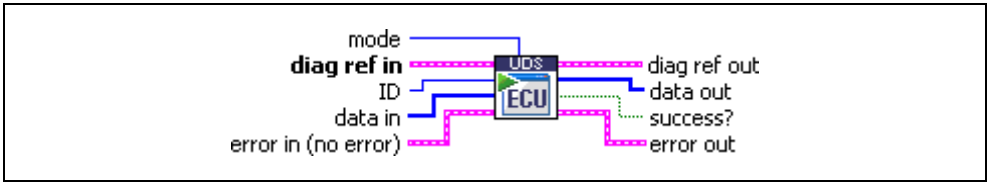
UDS RequestUpload.vi initiates the upload of a data block from the ECU. This is required to set up the upload process; the actual data transfer occurs with subsequent **UDS TransferData.vi** requests. The transfer must occur in blocks of the size that this service returns (the **block size** parameter). After the upload completes, use the **UDS RequestTransferExit.vi** service to terminate the process.

UDS RoutineControl.vi

Purpose

Executes the UDS RoutineControl service. Executes a routine on the ECU.

Format



Input

mode defines the service operation mode. You can obtain the values from a ring control:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the routine to be started. The values are application specific.

data in defines application-specific input parameters for the routine.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI executes the UDS RoutineControl service and launches an ECU routine, stops an ECU routine, or requests ECU routine results from the ECU.

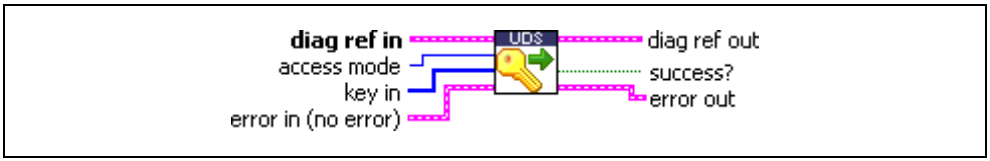
For further details about this service, refer to the ISO 15765-3 standard.

UDS SendKey.vi

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

access mode indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

key in defines the key data to be sent to the ECU.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The usual procedure for getting a security access to the ECU is as follows:

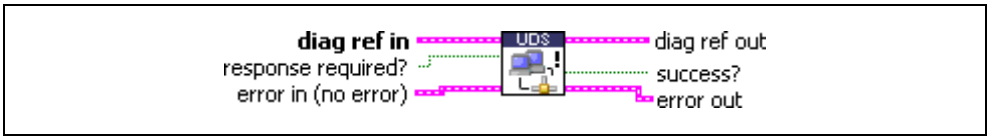
1. Request a seed from the ECU using **UDS RequestSeed.vi** with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using **UDS SendKey.vi** with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

UDS TesterPresent.vi

Purpose

Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

response required? indicates whether the ECU answers this service (TRUE, default) or not (FALSE). In the latter case, **success?** is TRUE.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

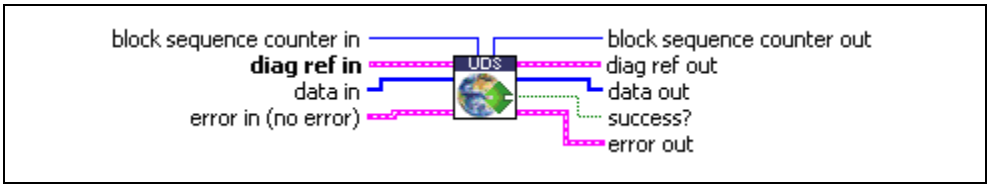
Keep calling **UDS TesterPresent.vi** within the ECU timeout period if no other service is executed.

UDS TransferData.vi

Purpose

Transfers data to/from the ECU in a download/upload process.

Format



Input

block sequence counter in is used to number the data blocks to be transferred to/from the ECU. The block sequence counter value starts at 01 hex with the first **UDS TransferData.vi** request that follows the **UDS RequestDownload.vi** or **UDS RequestUpload.vi** service. Its value is incremented by 1 for each subsequent **UDS TransferData.vi** request. At the value of FF hex, the block sequence counter rolls over and starts at 00 hex with the next **UDS TransferData.vi** request.

The block sequence counter is updated automatically and returned in the **block sequence counter out** parameter.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data in defines the data block to be written to the ECU.

For a download, this is a memory data block to be downloaded to the ECU.

For an upload, the meaning is implementation dependent; in most cases, it is sufficient to leave the parameter empty (default).

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

block sequence counter out returns the updated value of the block sequence counter.

diag ref out is a copy of **diag ref in**. It can be wired to subsequent diagnostic VIs.

data out returns the memory data from the ECU.

For a download, this might contain a checksum or similar verification instrument; the meaning is implementation dependent.

For an upload, this is a memory data block uploaded from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

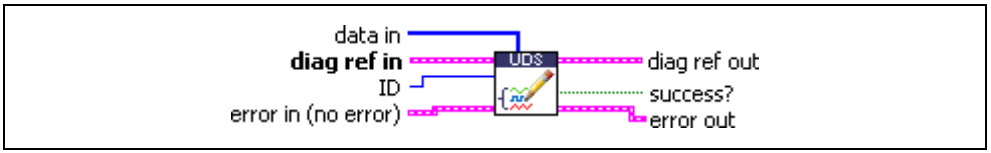
UDS TransferData.vi executes the data transfer of a download process (initiated with a previous **UDS RequestDownload.vi** request) or an upload process (initiated with a previous **UDS RequestUpload.vi** request). The data transfer must occur in blocks of the size returned in the **block size** parameter of the respective request service. After the data transfer has completed, terminate the operation by calling the **UDS RequestTransferExit.vi** service.

UDS WriteDataByIdentifier.vi

Purpose

Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.

Format



Input

data in defines the data record to be written to the ECU. If you know the the data description record, you can use **Convert from Phys.vi** to generate this record.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

ID defines the identifier of the data to be written. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the UDS service WriteDataByIdentifier and writes RecordValues (data values) to the ECU. **data in** identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

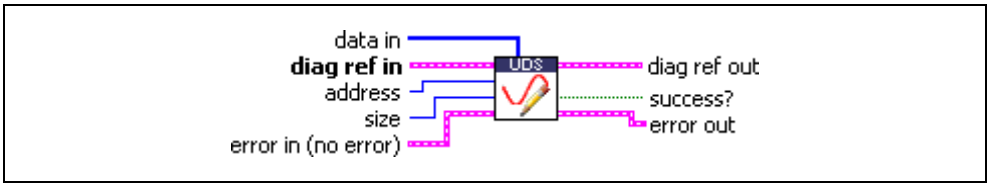
For further details about this service, refer to the 15765-3 standard.

UDS WriteMemoryByAddress.vi

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format



Input

data in defines the memory block to be written to the ECU.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

address defines the memory address to which data are to be written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

size defines the length of the memory block to be written.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI performs the UDS service WriteMemoryByAddress and writes RecordValues (data values) to the ECU. **address** and **size** identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

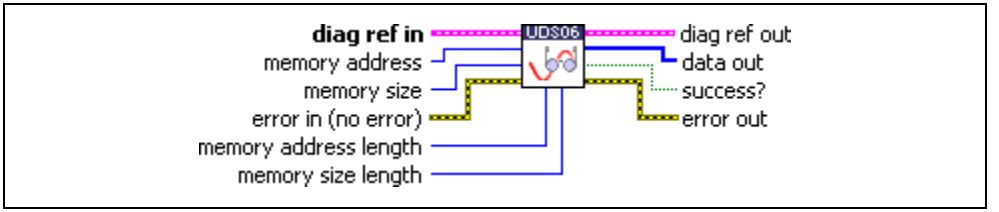
For further details about this service, refer to the ISO 15765-3 standard.

UDS06 ReadMemoryByAddress.vi

Purpose

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address from which data are to be read. Note that **memory address length** specifies how many bytes of the address are sent to the ECU. This defines the maximum address you can use.

memory size defines the length of the memory block to be read. Note that **memory size length** specifies how many bytes of the size are sent to the ECU. This defines the maximum size you can use.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length specifies how many bytes of the address are sent to the ECU. The default is 4.

memory size length specifies how many bytes of the size are sent to the ECU. The default is 4.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the memory data from the ECU.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

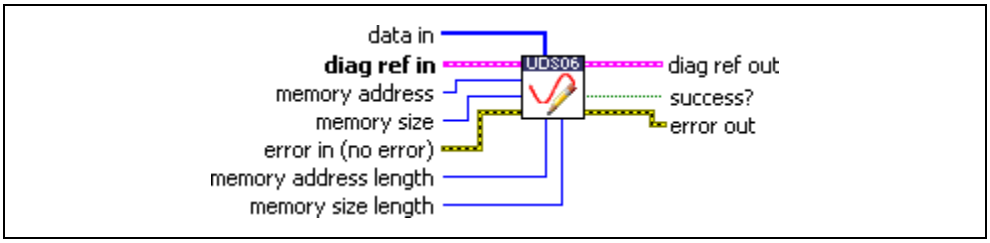
Similar to **UDS ReadMemoryByAddress.vi**. You can define the size in bytes of the address and size parameters (the default is 4).

UDS06 WriteMemoryByAddress.vi

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format



Input

data in defines the memory block to be written to the ECU.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

memory address defines the memory address to which data are to be sent. Note that **memory address length** specifies how many bytes of the address are sent to the ECU. This defines the maximum address you can use.

memory size defines the length of the memory block to be sent. Note that **memory size length** specifies how many bytes of the size are sent to the ECU. This defines the maximum size you can use.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to

a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

memory address length specifies how many bytes of the address are sent to the ECU. The default is 4.

memory size length specifies how many bytes of the size are sent to the ECU. The default is 4.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Similar to **UDS WriteMemoryByAddress.vi**. You can define the size in bytes of the address and size parameters (the default is 4).

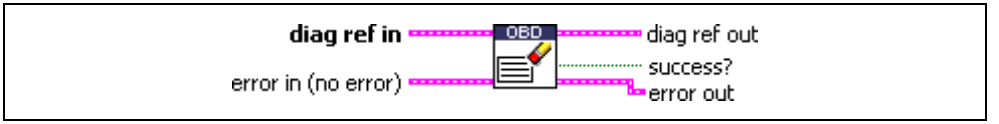
OBD (On-Board Diagnostics) Services

OBD Clear Emission Related Diagnostic Information.vi

Purpose

Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related Diagnostic Trouble Codes (DTCs) in the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

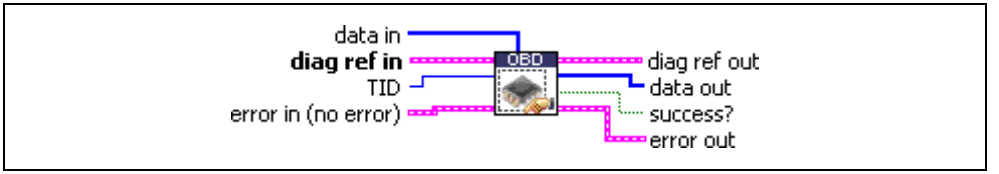
source identifies the VI where the error occurred.

OBD Request Control Of On-Board Device.vi

Purpose

Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.

Format



Input

data in defines application-specific data for this service.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

TID defines the test identifier of the I/O to be manipulated. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific data for this service.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

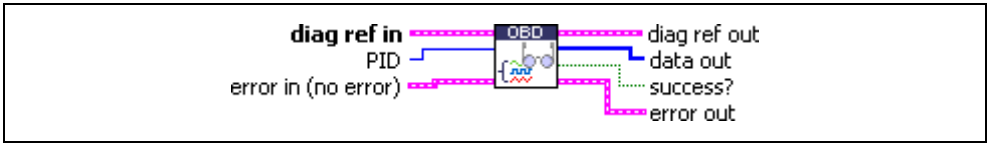
source identifies the VI where the error occurred.

OBD Request Current Powertrain Diagnostic Data.vi

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

PID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

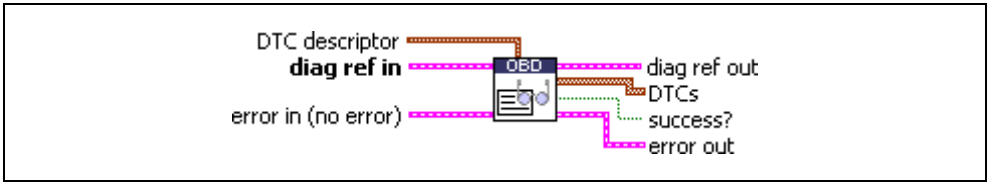
source identifies the VI where the error occurred.

OBD Request Emission Related DTCs.vi

Purpose

Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

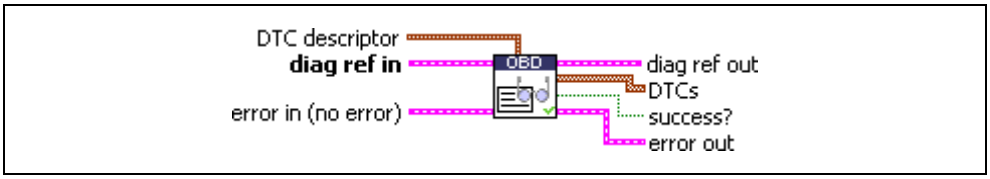
source identifies the VI where the error occurred.

OBD Request Emission Related DTCs During Current Drive Cycle.vi

Purpose

Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

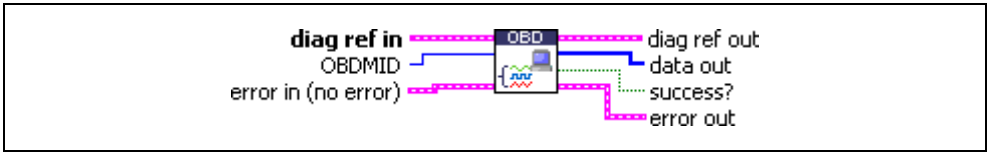
source identifies the VI where the error occurred.

OBD Request On-Board Monitoring Test Results.vi

Purpose

Executes the OBD Request On-Board Monitoring Test Results service. Reads a test data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

OBDMID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

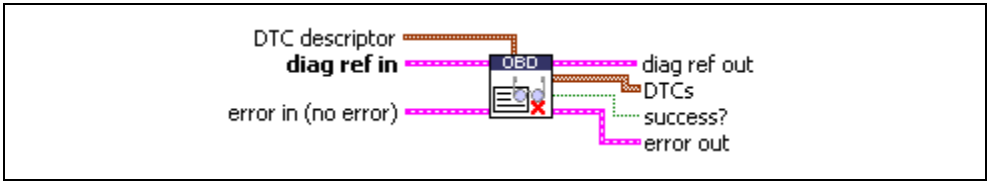
source identifies the VI where the error occurred.

OBD Request Permanent Fault Codes.vi

Purpose

Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 2.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

This VI interprets the response byte stream according to this description and returns the resulting DTC records in the **DTCs** cluster array.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, you can use **DTC to String.vi** to convert this to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

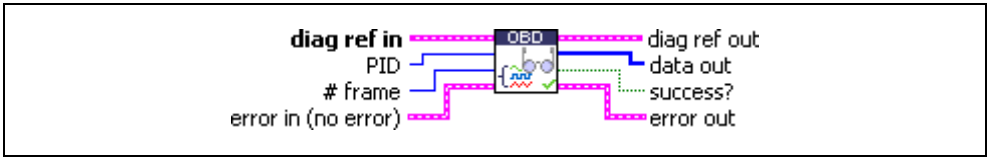
source identifies the VI where the error occurred.

OBD Request Powertrain Freeze Frame Data.vi

Purpose

Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a Diagnostic Trouble Code occurred.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

PID defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

 # frame is the number of the freeze frame from which the data are to be retrieved.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

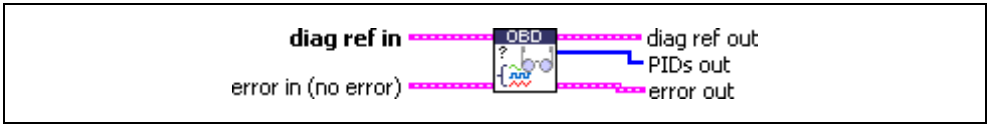
source identifies the VI where the error occurred.

OBD Request Supported PIDs.vi

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service to retrieve the valid PID values for this service.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

PIDs out returns an array of valid PIDs for the OBD Request Current Powertrain Diagnostic Data service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

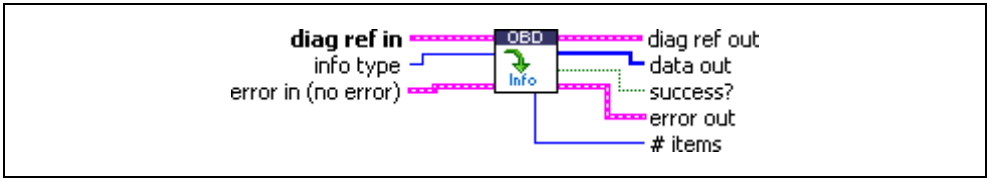
source identifies the VI where the error occurred.

OBD Request Vehicle Information.vi

Purpose

Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

info type defines the type of information to be read. The values are defined in the SAE J1979 standard.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the vehicle information from the ECU. You can obtain the description from the SAE J1979 standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

 # items is the number of data items (not bytes) this service returns.

WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) Services

WWH-OBD Clear Emission Related DTCs.vi

Purpose

Executes the WWH-OBD ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The WWH-OBD ClearDiagnosticInformation service is based on the UDS ClearDiagnosticInformation service (ISO 14229-1).

WWH-OBD Convert DTCs to J1939.vi

Purpose

Converts DTCs to the J1939 DTC format.

Format

Input

DTCs is a cluster that contains diagnostic trouble codes (DTCs).

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DTCs J1939 contains the DTCs converted to the J1939 format.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

WWH-OBD Convert DTCs to J2012.vi

Purpose

Converts DTCs to the J2012 DTC format.

Format

Input

DTCs is a cluster that contains diagnostic trouble codes (DTCs).

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DTCs J2012 contains the DTCs converted to the J2012 format.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

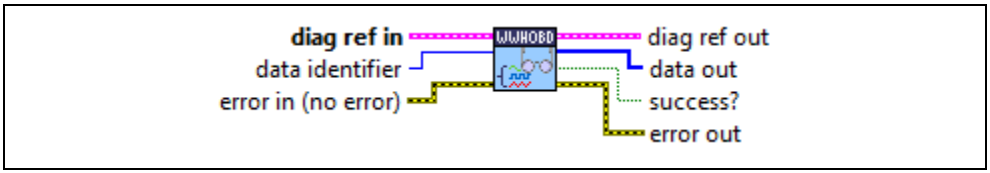
source identifies the VI where the error occurred.

WWH-OBD Request DID.vi

Purpose

Executes the WWH-OBD ReadDataByIdentifier service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

data identifier defines the data identifier of the data to be read. The SAE J1979DA standard defines the values.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record. If you know the record data description, you can use **Convert from Phys.vi** to interpret this record. You can obtain the description from the SAE J1979DA standard.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

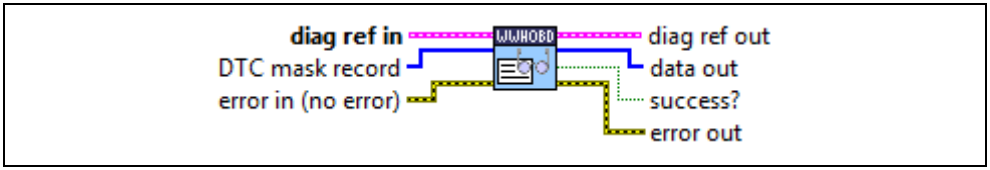
The WWH-OBD ReadDataByIdentifier service is based on the UDS ReadDataByIdentifier service (ISO 14229-1).

WWH-OBD Request DTC Extended Data Record.vi

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DTC mask record specifies the DTC mask record.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

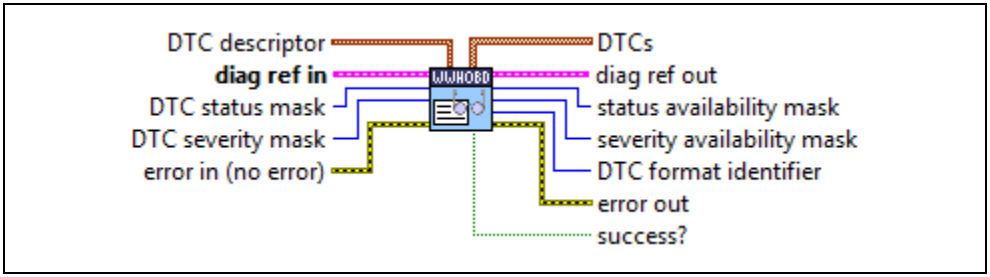
The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

WWH-OBD Request Emission Related DTCs.vi

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

DTC descriptor is a cluster that describes the DTC records the ECU delivers:

DTC Byte Length indicates the number of bytes the ECU sends for each DTC. The default is 3.

Status Byte Length indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

Add Data Byte Length indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there is no additional data, so the default is 0.

Byte Order indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola) (default)

1: LSB_FIRST (Intel)

The **DTC descriptor** is given here as a parameter to convert the group of DTC parameters to a binary representation according to **DTC Byte Length** and **Byte Order**.

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DTC status mask defines the status of DTCs to be read. The values are application specific.

DTC severity mask defines the severity information of DTCs to be read. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DTCs returns the resulting DTCs as an array of clusters:

DTC is the resulting Diagnostic Trouble Code. You can use **WWH-OBD Convert DTCs to J1939.vi** or **WWH-OBD Convert DTCs to J2012.vi** to convert this to readable format as SAE J1939 and SAE J2012 define.

Status is the DTC status. Usually, this is a bit field with the following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisOperationCycle
2	pendingDTC
3	confirmedDTC

4 testNotCompletedSinceLastClear

5 testFailedSinceLastClear

6 testNotCompletedThisOperationCycle

7 warningIndicatorRequested

For OBD, this field usually does not contain valid information.

Add Data contains optional additional data for this DTC. Usually, this does not contain valid information (refer to **DTC descriptor**).

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DTC status availability mask is an application-specific value returned for all DTCs.

DTC severity availability mask is an application-specific value returned for all DTCs.

DTC format identifier is an application-specific value returned for all DTCs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

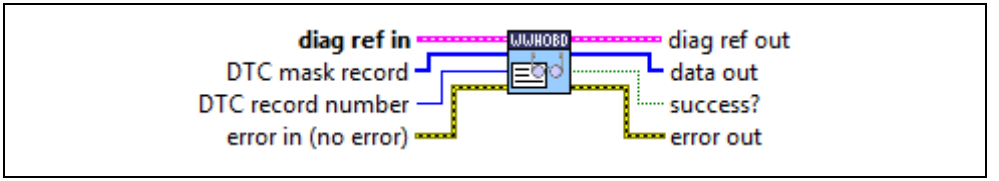
The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

WWH-OBD Request Freeze Frame Information.vi

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DTC mask record defines the DTC to be read. The values are application specific.

DTC record number specifies the snapshot record number.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns the ECU data record.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

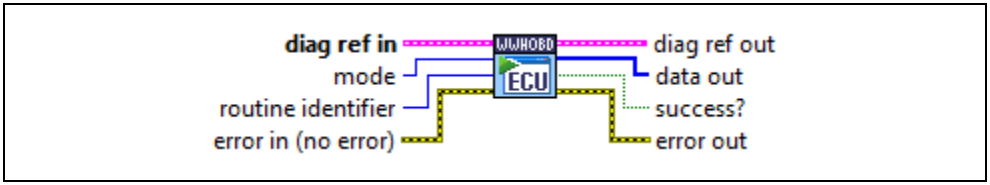
The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

WWH-OBD Request RID.vi

Purpose

Executes the WWH-OBD RoutineControl service. Reads a data record from the ECU.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

mode defines the service operation mode. You can obtain the values from a ring control:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

routine identifier defines the identifier of the routine to be started. The values are application specific.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning

is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

data out returns application-specific output parameters from the routine.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

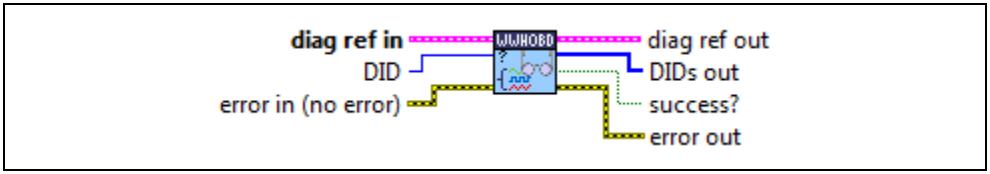
The WWH-OBD RoutineControl service is based on the UDS RoutineControl service (ISO 14229-1).

WWH-OBD Request Supported DIDs.vi

Purpose

Executes the WWH-OBD ReadDataByIdentifier service to retrieve the valid DID values for this service.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

DID specifies the diagnostic data identifier for this service. The following values are valid and can be obtained through an enum control:

0 **PID:** parameter identifier

1 **MID:** monitor identifier

2 **ITID:** info type identifier

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

DIDs out returns an array of valid DIDs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

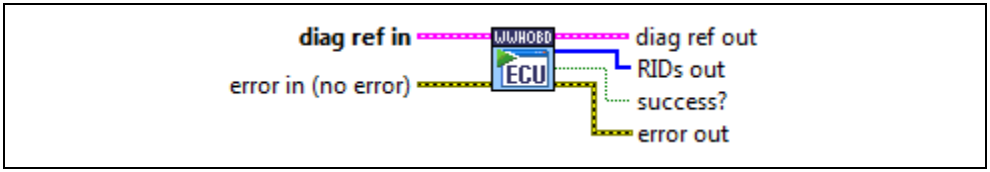
The WWH-OBD ReadDataByIdentifier service is based on the UDS ReadDataByIdentifier service (ISO 14229-1).

WWH-OBD Request Supported RIDs.vi

Purpose

Executes the WWH-OBD RoutineControl service to retrieve the valid RID values for this service.

Format



Input

diag ref in specifies the diagnostic session handle, obtained from **Open Diagnostic.vi** or **Open Diagnostic on IP.vi** and wired through subsequent diagnostic VIs. Normally, it is not necessary to manually manipulate the elements of this cluster.

error in is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI is not executed when status is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

diag ref out is a copy of **diag ref in**. You can wire it to subsequent diagnostic VIs.

RIDs out returns an array of valid RIDs.

success? indicates successful receipt of a positive response message for this diagnostic service.

error out describes error conditions. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: the VI did not execute the intended operation. A positive value means warning: the VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

The WWH-OBD RoutineControl service is based on the UDS RoutineControl service (ISO 14229-1).

6

Automotive Diagnostic Command Set API for C

This chapter lists the Automotive Diagnostic Command Set API functions and describes their format, purpose, and parameters. Unless otherwise stated, each Automotive Diagnostic Command Set function suspends execution of the calling thread until it completes. The functions are listed alphabetically in four categories: general functions, KWP2000 services, UDS (DiagOnCAN) services, and OBD (On-Board Diagnostics) services.

Section Headings

The following are section headings found in the Automotive Diagnostic Command Set for C functions.

Purpose

Each function description includes a brief statement of the function purpose.

Format

The format section describes the function format for the C programming language.

Input and Output

The input and output sections list the function parameters.

Description

The description section gives details about the function purpose and effect.

List of Data Types

The following data types are used with the Automotive Diagnostic Command Set API for C functions.

Table 6-1. Data Types for the Automotive Diagnostic Command Set for C

Data Type	Purpose
`i8`	8-bit signed integer
`i16`	16-bit signed integer
`i32`	32-bit signed integer
`u8`	8-bit unsigned integer
`u16`	16-bit unsigned integer
`u32`	32-bit unsigned integer
`f32`	32-bit floating-point number
`f64`	64-bit floating-point number
`str`	ASCII string represented as an array of characters terminated by null character (`'\0'`). This type is used with output strings. `str` is typically used in the Automotive Diagnostic Command Set API as a pointer to a string, as `char*`.
`cstr`	ASCII string represented as an array of characters terminated by null character (`'\0'`). This type is used with input strings. `cstr` is typically used in the Automotive Diagnostic Command Set as a pointer to a string, as `const char*`.

List of Functions

The following table contains an alphabetical list of the Automotive Diagnostic Command Set API functions.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C

Function	Purpose
ndClearDiagnosticInformation	Executes the ClearDiagnostic Information service. Clears selected Diagnostic Trouble Codes (DTCs).
ndCloseDiagnostic	Closes a diagnostic session.
ndControlDTCSetting	Executes the ControlDTCSetting service. Modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).
ndConvertFromPhys	Converts a physical data value into a binary representation using a type descriptor.
ndConvertToPhys	Converts a binary representation of a value into its physical value using a type descriptor.
ndCreateExtendedCANIds	Creates diagnostic CAN identifiers according to ISO 15765-2.
ndDiagFrameRecv	Receives a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.
ndDiagFrameSend	Sends a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndDiagnosticService	Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this function.
ndDisableNormalMessageTransmission	Executes the DisableNormalMessage Transmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).
ndDoIPActivateRouting	Defines the source and target address for a DoIP TCP/IP connection.
ndDoIPConnect	Creates a TCP/IP connection to a DoIP entity identified by its IP address.
ndDoIPDisconnect	Disconnects the TCP/IP connection to a DoIP entity.
ndDoIPEntityStatus	Gets status information from a DoIP entity.
ndDoIPGetDiagPowerMode	Gets information on the DoIP entity power state.
ndDoIPGetEntities	Returns a table of all DoIP entities (vehicles) on the local subnet, possibly restricted to EID or VIN.
ndDoIPSendVehicleIdentRequest	Sends a UDP request to all DoIP-capable vehicles in the local subnet to identify themselves.
ndDoIPSendVehicleIdentReqEID	Sends a UDP request to all DoIP-capable vehicles with a certain EID (MAC address) in the local subnet to identify themselves.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndDoIPSendVehicleIdentReqVIN	Sends a UDP request to all DoIP-capable vehicles with a certain VIN (Vehicle Identification Number) in the local subnet to identify themselves.
ndDTCToString	Returns a string representation (such as *P1234*) for a 2-byte diagnostic trouble code (DTC).
ndECUReset	Executes the ECUReset service. Resets the ECU.
ndEnableNormalMessageTransmission	Executes the EnableNormalMessage Transmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).
ndGetProperty	Gets a diagnostic global internal parameter.
ndGetTimeStamp	Gets timestamp information about the first/last send/received frame of the ISO TP for CAN and LIN.
ndInputOutputControlByLocalIdentifier	Executes the InputOutputControlBy LocalIdentifier service. Modifies the ECU I/O port behavior.
ndOBDClearEmissionRelatedDiagnosticInformation	Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related diagnostic trouble codes (DTCs) in the ECU.
ndOBDOpen	Opens a diagnostic session on a CAN port for OBD-II.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndOBDRequestControlOfOnBoardDevice	Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.
ndOBDRequestCurrentPowertrainDiagnosticData	Executes the OBD Request Current Powertrain Diagnostic Data service. Reads an ECU data record.
ndOBDRequestEmissionRelatedDTCs	Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).
ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle	Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.
ndOBDRequestOnBoardMonitoringTestResults	Executes the OBD Request On-Board Monitoring Test Results service. Reads an ECU test data record.
ndOBDRequestPermanentFaultCodes	Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.
ndOBDRequestPowertrainFreezeFrameData	Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a diagnostic trouble code occurred.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndOBDRequestVehicleInformation	Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.
ndOpenDiagnostic	Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.
ndOpenDiagnosticOnIP	Opens a diagnostic session on an IP port. Communication to the ECU is not yet started.
ndOpenDiagnosticOnLIN	Opens a diagnostic session on an NI-XNET LIN port. Communication to the ECU is not yet started.
ndReadDataByLocalIdentifier	Executes the ReadDataByLocal Identifier service. Reads an ECU data record.
ndReadDTCByStatus	Executes the ReadDiagnosticTrouble CodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).
ndReadECUIdentification	Executes the ReadECUIdentification service. Returns ECU identification data from the ECU.
ndReadMemoryByAddress	Executes the ReadMemoryByAddress service. Reads data from the ECU memory.
ndReadStatusOfDTC	Executes the ReadStatusOfDiagnostic TroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndRequestRoutineResultsByLocalIdentifier	Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from an ECU routine.
ndRequestSeed	Executes the SecurityAccess service to retrieve a seed from the ECU.
ndSendKey	Executes the SecurityAccess service to send a key to the ECU.
ndSetProperty	Set a diagnostic global internal parameter.
ndStartDiagnosticSession	Executes the StartDiagnosticSession service. The ECU is set up in a specific diagnostic mode.
ndStartRoutineByLocalIdentifier	Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.
ndStatusToString	Returns a description for an error code.
ndStopDiagnosticSession	Executes the StopDiagnosticSession service. Returns the ECU to normal mode.
ndStopRoutineByLocalIdentifier	Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.
ndTesterPresent	Executes the TesterPresent service. Keeps the ECU in diagnostic mode.
ndUDS06ReadMemoryByAddress	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndUDS06WriteMemoryByAddress	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
ndUDSClearDiagnosticInformation	Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).
ndUDSCommunicationControl	Executes the UDS CommunicationControl service. Switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off.
ndUDSControlDTCSetting	Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) behavior.
ndUDSDiagnosticSessionControl	Executes the UDS DiagnosticSessionControl service. The ECU is set up in a specific diagnostic mode.
ndUDSECUReset	Executes the UDS ECUReset service. Resets the ECU.
ndUDSInputOutputControlByIdentifier	Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.
ndUDSReadDataByIdentifier	Executes the UDS ReadDataByIdentifier service. Reads an ECU data record.
ndUDSReadMemoryByAddress	Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndUDSReportDTCBySeverityMaskRecord	Executes the ReportDTCBySeverity MaskRecord subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndUDSReportDTCByStatusMask	Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndUDSReportSeverityInformationOfDTC	Executes the ReportSeverity InformationOfDTC subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs) are read.
ndUDSReportSupportedDTCs	Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTrouble CodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).
ndUDSRequestDownload	Initiates a download of data to the ECU.
ndUDSRequestSeed	Executes the UDS SecurityAccess service to retrieve a seed from the ECU.
ndUDSRequestTransferExit	Terminates a download/upload process.
ndUDSRequestUpload	Initiates an upload of data from the ECU.

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndUDSRoutineControl	Executes the UDS RoutineControl service. Executes a routine on the ECU.
ndUDSSendKey	Executes the UDS SecurityAccess service to send a key to the ECU.
ndUDSTesterPresent	Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.
ndUDSTransferData	Transfers data to/from the ECU in a download/upload process.
ndUDSWriteDataByIdentifier	Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.
ndUDSWriteMemoryByAddress	Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.
ndVWTPConnect	Establishes a connection channel to an ECU using the VW TP 2.0.
ndVWTPConnectionTest	Maintains a connection channel to an ECU using the VW TP 2.0.
ndVWTPDisconnect	Terminates a connection channel to an ECU using the VW TP 2.0.
ndWriteDataByLocalIdentifier	Executes the WriteDataByLocal Identifier service. Writes a data record to the ECU.
ndWriteMemoryByAddress	Executes the WriteMemoryByAddress service. Writes data to the ECU memory.
ndWWHOBDClearEmissionRelatedDTCs	Executes the WWH-OBD ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Table 6-2. Functions for the Automotive Diagnostic Command Set for C (Continued)

Function	Purpose
ndWWHOBDConvertDTCsToJ1939	Converts DTCs to the J1939 DTC format.
ndWWHOBDConvertDTCsToJ2012	Converts DTCs to the J2012 DTC format.
ndWWHOBDRequestDID	Executes the WWH-OBD ReadDataByIdentifier service. Reads a data record from the ECU.
ndWWHOBDRequestDTCExtendedDataRecord	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndWWHOBDRequestEmissionRelatedDTCs	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndWWHOBDRequestFreezeFrameInformation	Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).
ndWWHOBDRequestRID	Executes the WWH-OBD RoutineControl service. Reads a data record from the ECU.
ndWWHOBDRequestSupportedDIDs	Executes the WWH-OBD ReadDataByIdentifier service to retrieve the valid DID values for this service.
ndWWHOBDRequestSupportedRIDs	Executes the WWH-OBD RoutineControl service to retrieve the valid RID values for this service.

General Functions

ndCloseDiagnostic

Purpose

Closes a diagnostic session.

Format

```
long ndCloseDiagnostic(
       TD1 *diagRefIn);
```

Input

`diagRefIn`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The diagnostic session `diagRefIn` specifies is closed, and you can no longer use it for communication to an ECU. This command does not communicate the closing to the ECU before terminating; if this is necessary, you must manually do so (for example, by calling `ndStopDiagnosticSession`) before calling `ndCloseDiagnostic`.

ndConvertFromPhys

Purpose

Converts a physical data value into a binary representation using a type descriptor.

Format

```
void ndConvertFromPhys(
       TD2 *typeDescriptor,
       double value,
       unsigned char dataOut[],
       long *len);
```

Input

`typeDescriptor`

A struct that specifies the conversion of the physical value to its binary representation:

```
typedef struct {
    long StartByte;
    long ByteLength;
    unsigned short ByteOrder;
    unsigned short DataType;
    double ScaleFactor;
    double ScaleOffset;
    } TD2;
```

`StartByte` is ignored by `ndConvertFromPhys`.

`ByteLength` is the number of bytes in the binary representation.

`ByteOrder` defines the byte order for multibyte representations. The values are:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

`DataType` is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths 4 or 8 are allowed.

`ScaleFactor` defines the physical value scaling:

Phys = (`ScaleFactor`) * (binary representation) + (`ScaleOffset`)

`ScaleOffset` (refer to `ScaleFactor`)

`value`

The physical value to be converted to a binary representation.

Output

`dataOut`

Points to the byte array to be filled with the binary representation of `value`.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

Description

Data input to diagnostic services (for example, `ndWriteDataByLocalIdentifier`) is usually a byte array of binary data. If you have the data input description (for example, *byte 3 and 4 are engine RPM scaled as .25 * × RPM in Motorola representation*), you can use `ndConvertFromPhys` to convert the physical value to the byte stream by filling an appropriate `typeDescriptor` struct.

`ndConvertFromPhys` converts only the portion specified by one type descriptor to a binary representation. If your data input consists of several values, you can use `ndConvertFromPhys` multiple times on different parts of the byte array.

ndConvertToPhys

Purpose

Converts a binary representation of a value into its physical value using a type descriptor.

Format

```
void ndConvertToPhys(
       TD2 *typeDescriptor,
       unsigned char dataIn[],
       long len,
       double *value);
```

Input

`typeDescriptor`

A struct that specifies the conversion of the physical value to its binary representation:

```
typedef struct {
    long StartByte;
    long ByteLength;
    unsigned short ByteOrder;
    unsigned short DataType;
    double ScaleFactor;
    double ScaleOffset;
    } TD2;
```

`StartByte` gives the start byte of the binary representation in the `dataIn` record.

`ByteLength` is the number of bytes in the binary representation.

`ByteOrder` defines the byte order for multibyte representations. The values are:

0: MSB_FIRST (Motorola)

1: LSB_FIRST (Intel)

`DataType` is the binary representation format:

0: Unsigned. Only byte lengths of 1–4 are allowed.

1: Signed. Only byte lengths of 1–4 are allowed.

2: Float. Only byte lengths 4 or 8 are allowed.

`ScaleFactor` defines the physical value scaling:

Phys = (`ScaleFactor`) * (binary representation) + (`ScaleOffset`)

`ScaleOffset` (refer to `ScaleFactor`)

`dataIn`

Points to the byte array that contains the binary representation of `value`.

`len`

Must contain the `dataIn` array length.

Output

`value`

The physical value converted from the binary representation.

Description

Data output from diagnostic services (for example, `ndReadDataByLocalIdentifier`) is usually a byte stream of binary data. If you have a description of the data output (for example, *byte 3 and 4 are engine RPM scaled as .25 *× RPM in Motorola representation*), you can use `ndConvertToPhys` to extract the physical value from the byte stream by filling an appropriate `typeDescriptor` struct.

ndCreateExtendedCANIds

Purpose

Creates diagnostic CAN identifiers according to ISO 15765-2.

Format

```
void ndCreateExtendedCANIds (
       unsigned short addressingMode,
       unsigned short transportProtocol,
       unsigned char sourceAddress,
       unsigned char targetAddress,
       unsigned long *transmitID,
       unsigned long *receiveID);
```

Input

`addressingMode`

Specifies whether the ECU is physically or functionally addressed:

0: physical addressing

1: functional addressing

`transportProtocol`

Specifies whether normal or mixed mode addressing is used. The following values are valid:

0 **ISO TP—Normal Mode**. The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**. The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

`sourceAddress`

The host (diagnostic tester) logical address.

`targetAddress`

The ECU logical address.

Output

`transmitID`

The generated CAN identifier for sending diagnostic request messages from the host to the ECU.

`receiveID`

The generated CAN identifier for sending diagnostic response messages from the ECU to the host.

Description

ISO 15765-2 specifies a method for creating (extended/29 bit) CAN identifiers for diagnostic applications given the addressing mode (physical/functional), the transport protocol (normal/mixed), and the 8-bit source and target addresses. This function implements the construction of these CAN identifiers. You can use them directly in the `ndOpenDiagnostic` function.

ndDiagFrameRecv

Purpose

Receives a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.

Format

```
long ndDiagFrameRecv(
       TD1 *diagRef,
       unsigned long timeout,
       unsigned char dataOut[],
       long *len);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`timeout`

Specifies the time to wait for the arrival of a message. If no message arrives within this time, a timeout error is returned.

Output

`dataOut`

Returns up to 8 bytes of payload data from a CAN frame received on the diagnostic identifier.

`len`

On input, `len` must contain the number of bytes provided for the `dataOut` buffer. On output, it returns the number of valid data bytes in `dataOut`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDiagFrameRecv` receives an arbitrary raw CAN frame on the diagnostic CAN identifier. For example, you can check the transport protocol implementation of an ECU for correct responses if erroneous protocol requests are issued.

ndDiagFrameSend

Purpose

Sends a raw CAN frame on the diagnostic CAN ID to check for errors in the transport protocol implementation of an ECU.

Format

```
long ndDiagFrameSend(
       TD1 *diagRef,
       unsigned char dataIn[],
       long len);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`dataIn`

An array of up to 8 bytes sent as a CAN payload on the diagnostic identifier.

`len`

Must contain the number of (valid) data bytes in `dataIn`.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDiagFrameSend` transmits an arbitrary raw CAN frame on the diagnostic CAN identifier. For example, you can check the transport protocol implementation of an ECU for robustness if erroneous protocol requests are issued.

ndDiagnosticService

Purpose

Executes a generic diagnostic service. If a special service is not available through the KWP2000, UDS, or OBD service functions, you can build it using this function.

Format

```
long ndDiagnosticService(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, `dataOut` returns no values, and `len2` returns 0. This parameter is passed by reference.

`dataIn`

Contains the request message byte sequence for the diagnostic service sent to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Contains the response message byte sequence of the diagnostic service returned from the ECU.

`len2`

On input, `len2` must contain the number of bytes provided for the `dataOut` buffer. On output, it returns the number of valid data bytes in `dataOut`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDiagnosticService` is a generic routine to execute any diagnostic service. The request and response messages are fed unmodified to the `dataIn` input and retrieved from the `dataOut` output, respectively. No interpretation of the contents is done, with one exception: The error number is retrieved from a negative response, if one occurs. In this case, an error is communicated through the return value.

All specialized diagnostic services call `ndDiagnosticService` internally.

ndDTCToString

Purpose

Returns a string representation (such as *P1234*) for a 2-byte diagnostic trouble code (DTC).

Format

```
void ndDTCToString(
       unsigned long DTCNum,
       char DTCString[],
       long *len);
```

Input

`DTCNum`

The DTC number as returned in the DTCs structs of `ndReadDTCByStatus`, `ndReadStatusOfDTC`, `ndUDSReportDTCBySeverityMaskRecord`, `ndUDSReportDTCByStatusMask`, `ndUDSReportSeverityInformationOfDTC`, `ndUDSReportSupportedDTCs`, `ndOBDRequestEmissionRelatedDTCs`, or `ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle`.

Note This function converts only 2-byte DTCs. If you feed in larger numbers, the function returns garbage.

Output

`DTCString`

The DTC string representation.

`len`

On input, `len` must contain the `DTCString` array length (at least 6). On return, it contains the number of valid data bytes in the `DTCString` array.

Description

The SAE J2012 standard specifies a naming scheme for 2-byte DTCs consisting of one letter and four digits. Use `ndDTCToString` to convert the DTC numerical representation to this name.

ndGetProperty

Purpose

Gets a diagnostic global internal parameter.

Format

```
uint32_t ndGetProperty(uint16_t propertyID);
```

Input

`propertyID`

Defines the parameter whose value is to be retrieved:

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms for CAN and 1000 ms for LIN.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If you set this value to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** returns whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** returns the CAN frame content if filled with defined data or random data bytes.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** returns how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by `success` = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** is the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

13 **STmin** is the minimum time in seconds between the end of transmission of a frame in a diagnostic request message and the start of transmission of the next frame in the diagnostic request message for LIN-based diagnostic communication. The default is 0.

14 **P2min** is the minimum time in seconds between reception of the last frame of the diagnostic request and the response sent by the node for LIN-based diagnostic communication. The default is 0.05.

15 **Termination** reads the NI-XNET Termination property. Reflections on the CAN and LIN bus can cause communication failures. To prevent reflections, termination can be present as external resistance or resistance the XNET CAN or LIN board applies internally. This property determines whether the XNET board uses termination to the bus. For further information about appropriate terminations of a CAN or LIN network, refer to the *NI-XNET Hardware and Software Manual*. The default is 0.

Output

`propertyValue`

The requested property value.

Description

Use this function to request several internal diagnostic parameters, such as timeouts for the transport protocol. Use `ndSetProperty` to modify the parameters.

ndGetTimeStamp

Purpose

Gets timestamp information about the first/last send/received frame of the ISO TP for CAN and LIN.

Format

```
void ndGetTimeStamp (
       TD1 *diagRef,
       unsigned long long *timeStampWriteFirst,
       unsigned long long *timeStampWriteLast,
       unsigned long long *timeStampReadFirst,
       unsigned long long *timeStampReadLast);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnLIN` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`timeStampWriteFirst`

Contains the timestamp of the first write frame. This is usually the FF or SF of the ISO TP.

`timeStampWriteLast`

Contains the timestamp of the last write frame. This is usually the CF or SF of the ISO TP.

`timeStampReadFirst`

Contains the timestamp of the first read frame. This is usually the FF or SF of the ISO TP.

`timeStampReadLast`

Contains the timestamp of the last read frame. This is usually the CF or SF of the ISO TP.

Description

Use this function to get the first and last write CAN or LIN frame and the first and last read CAN or LIN frame if the ISO TP transport protocol is used. For all other transport protocols, the timestamps are always 0.

The received timestamps should be converted to system time using the FileTimeToLocalFileTime and FileTimeToSystemTime functions. Add `<windows.h>` to your project for this.

The UDS Get DTCs example includes an example for getting the timestamp.

ndOBDOpen

Purpose

Opens a diagnostic session on a CAN port for OBD-II.

Format

long ndOBDOpen (

```
char CANInterface[],
unsigned long baudrate,
unsigned long transmitID,
unsigned long receiveID,
TD1 *diagRefOut);
```

Input

`CANInterface`

Specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@nixnet* to the protocol string (for example, *CAN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

Note By selecting *nixnet* as the interface string, the Automotive Diagnostic Command Set uses the Frame Input and Output Queued sessions. To force the use of Frame Input and Output Stream sessions instead, select *ni_genie_nixnet* as the interface string (for example, *CAN1@ni_genie_nixnet*). An application instance can use only one Frame Input Stream Session and one Frame Output Stream Session at a time, so use the default name *nixnet* as the interface string, so that multiple NI-XNET Frame Queued Sessions can coexist on a single interface, and the Frame Input and Output Stream Sessions may be used, for example, for a Frame logging/replay use case.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

`baudrate`

The diagnostic communication baud rate.

`transmitID`

The CAN identifier for sending diagnostic request messages from the host to the ECU. Set to –1 (0xFFFFFFFF) for the default OBD CAN identifier.

`receiveID`

The CAN identifier for sending diagnostic response messages from the ECU to the host. Set to –1 (0xFFFFFFFF) for the default OBD CAN identifier.

Output

`diagRefOut`

A struct containing all necessary information about the diagnostic session. This is passed as a handle to all subsequent diagnostic functions, and you must close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndOBDOpen` opens a diagnostic communication channel to an ECU for OBD-II. The CAN port specified as input is initialized, and a handle to it is stored (among other internal data) into the `diagRefOut` struct, which serves as reference for further diagnostic functions.

If the `transmitID` and `receiveID` parameters are set to –1, communication is first tried on the default 11-bit OBD CAN identifiers; if that fails, the default 29-bit OBD CAN identifiers are tried. If that also fails, an error is returned.

If valid `transmitID` or `receiveID` parameters (11-bit or 29-bit with bit 29 set) are given, communication is tried on these identifiers. If that fails, an error is returned.

In general, it is not necessary to manipulate the `diagRefOut` struct contents.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware with NI-XNET Frame Input/Output-based sessions:

- **CAN1@nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using NI-XNET hardware with NI-XNET Stream Input/Output-based sessions:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1, c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO:

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

ndOpenDiagnostic

Purpose

Opens a diagnostic session on a CAN port. Communication to the ECU is not yet started.

Format

```
long ndOpenDiagnostic(
       char CANInterface[],
       unsigned long baudrate,
       unsigned short transportProtocol,
       unsigned long transmitID,
       unsigned long receiveID,
       TD1 *diagRefOut);
```

Input

`CANInterface`

Specifies the CAN interface on which the diagnostic communication should take place.

NI-CAN

The CAN interface is the name of the NI-CAN Network Interface Object to configure. This name uses the syntax *CANx*, where *x* is a decimal number starting at 0 that indicates the CAN network interface (CAN0, CAN1, up to CAN63). CAN network interface names are associated with physical CAN ports using Measurement and Automation Explorer (MAX).

NI-XNET

By default, the Automotive Diagnostic Command Set uses NI-CAN for CAN communication. This means you must define an NI-CAN interface for your NI-XNET hardware (NI-CAN compatibility mode) to use your XNET hardware for CAN communication. However, to use your NI-XNET interface in the native NI-XNET mode (meaning it does not use the NI-XNET Compatibility Layer), you must define your interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use. To do this, add *@nixnet* to the Protocol string (for example, *CAN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

Note By selecting *nixnet* as the interface string, the Automotive Diagnostic Command Set uses the Frame Input and Output Queued sessions. To force the use of Frame Input and Output Stream sessions instead, select *ni_genie_nixnet* as the interface string (for example, *CAN1@ni_genie_nixnet*). An application instance can use only one Frame Input Stream Session and one Frame Output Stream Session at a time, so use the default name *nixnet* as the interface string, so that multiple NI-XNET Frame Queued Sessions can coexist on a

single interface, and the Frame Input and Output Stream Sessions may be used, for example, for a Frame logging/replay use case.

CompactRIO or R Series

If using CompactRIO or R Series hardware, you must provide a bitfile that handles the CAN communication between the host system and FPGA. To access the CAN module on the FPGA, you must specify the bitfile name after the @ (for example, *CAN1@MyBitfile.lvbitx*). To specify a special RIO target, you can specify that target by its name followed by the bitfile name (for example, *CAN1@RIO1,MyBitfile.lvbitx*). Currently, only a single CAN interface is supported. RIO1 defines the RIO target name as defined in your LabVIEW Project definition. The *lvbitx* filename represents the filename and location of the bitfile on the host if using RIO or on a CompactRIO target. This implies that you must download the bitfile to the CompactRIO target before you can run your application. You may specify an absolute path or a path relative to the root of your target for the bitfile.

`baudrate`

The diagnostic communication baud rate.

`transportProtocol`

Specifies the transport protocol for transferring the diagnostic service messages over the CAN network. The following values are valid:

0 **ISO TP—Normal Mode**. The ISO TP as specified in ISO 15765-2 is used; all eight data bytes of the CAN messages are used for data transfer.

1 **ISO TP—Mixed Mode**. The ISO TP as specified in ISO 15765-2 is used; the first data byte is used as address extension.

2 **VW TP 2.0**.

`transmitID`

The CAN identifier for sending diagnostic request messages from the host to the ECU.

`receiveID`

The CAN identifier for sending diagnostic response messages from the ECU to the host.

Output

`diagRefOut`

A struct containing all necessary information about the diagnostic session. This is passed as a handle to all subsequent diagnostic functions, and you must close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndOpenDiagnostic` opens a diagnostic communication channel to an ECU. This function initializes the CAN port specified as input and stores a handle to it (among other internal data) into `diagRefOut`, which serves as reference for further diagnostic functions.

No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call `ndStartDiagnosticSession` or `ndUDSDiagnosticSessionControl`.

In general, you do not need to manipulate the `diagRefOut` struct contents, except if you use the **ISO TP—Mixed Mode** transport protocol, in which case you must store the address extensions for transmit and receive in the appropriate members of that struct.

Possible examples of selections for the interface parameter for the various hardware targets are as follows.

Using NI-CAN hardware:

- **CAN0**—uses CAN interface 0.
- **CAN1**—uses CAN interface 1 and so on with the form *CANx*.
- **CAN256**—uses virtual NI-CAN interface 256.

Using NI-XNET hardware with NI-XNET Frame Input/Output-based sessions:

- **CAN1@nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using NI-XNET hardware with NI-XNET Stream Input/Output-based sessions:

- **CAN1@ni_genie_nixnet**—uses CAN interface 1 of an NI-XNET device.
- **CAN2@ni_genie_nixnet**—uses CAN interface 2 of an NI-XNET device and so on with the form *CANx*.

Using R Series:

- **CAN1@RIO1, c:\temp\MyFpgaBitfile.lvbitx**—uses a named target RIO1 as compiled into the bitfile at location `c:\temp\MyFpgaBitfile.lvbitx`.

Using CompactRIO

- **CAN1@ \MyFpgaBitfile.lvbitx**—uses compiled bitfile `MyFpgaBitfile.lvbitx`, which must be FTP copied to the root of the CompactRIO target.

ndOpenDiagnosticOnIP

Purpose

Opens a diagnostic session on an IP port. Communication to the ECU is not yet started.

Format

```
long ndOpenDiagnosticOnIP(
       LVBoolean *dynamicPort,
       TD1 *diagRefOut);
```

Input

`dynamicPort`

Defines whether the standard UDP port 13401 (UDP_TEST_EQUIPMENT_LISTEN) is used for communication (FALSE) or a dynamically assigned UDP port (UDP_TEST_EQUIPMENT_REQUEST) is opened (TRUE).

Output

`diagRefOut`

A struct that contains all necessary information about the diagnostic session. Pass this struct as a handle to all subsequent diagnostic functions and close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

ndOpenDiagnosticOnIP opens a Diagnostic on Internet Protocol (DoIP) communication channel to an ECU. The UDP port specified as input is initialized, and a handle to it is stored (among other internal data) in diagRefOut, which serves as reference for further diagnostic functions.

Note that no communication to an ECU takes place at this point. To open a diagnostic session on an ECU, call ndDoIPGetEntities to find out which DoIP entities (DoIP-capable ECUs) exist in the network. You need to create a TCP/IP connection to the selected DoIP entity using ndDoIPConnect. After that, you can execute diagnostic services on the TCP/IP connection.

This VI replaces the standard (CAN-based) ndOpenDiagnostic, because the CAN parameters are no longer relevant for IP-based diagnostics.

In general, it is not necessary to manipulate the diagRefOut cluster contents.

ndOpenDiagnosticOnLIN

Purpose

Opens a diagnostic session on an NI-XNET LIN port. Communication to the ECU is not yet started.

Format

```
long ndOpenDiagnosticOnLIN(
       char LINInterface[],
       unsigned long baudrate,
       uint8_t NAD,
       char MasterReqFrame[],
       char SlaveRespFrame[]
       TD1 *diagRefOut);
```

Input

`LINInterface`

Specifies the NI-XNET LIN interface on which the diagnostic communication should take place, and selects the LIN Cluster name of a registered XNET Alias.

The Automotive Diagnostic Command Set supports NI-XNET hardware for LIN communication only. To use your NI-XNET LIN interface, you must define your LIN interface under **NI-XNET Devices** in MAX and pass the NI-XNET interface name that the Automotive Diagnostic Command Set will use.

To do this, add *@nixnet* to the interface string (for example, *LIN1@nixnet*). The interface name is related to the NI-XNET hardware naming under **Devices and Interfaces** in MAX.

The Automotive Diagnostic Command Set requires valid assignments to a LIN database such as LDF or FIBEX. To communicate with hardware products on the external network, applications must understand how that hardware communicates in the actual embedded system, such as the vehicle. This embedded communication is described within a standardized file, such as FIBEX (`.xml`) or LDF (`.ldf`) for LIN. Within NI-XNET, this file is referred to as a database. The database contains many object classes, each of which describes a distinct entity in the embedded system.

For LIN, you can select a LIN database and cluster to assign all settings automatically from the selected cluster, such as the LIN Baudrate.

Using NI-XNET hardware, the Interface string should look like the following examples:

- **LIN1@nixnet: XNET_LIN_Database**—Uses LIN interface 1 of an NI-XNET device and assigns the properties such as `baudrate` automatically from the XNET alias XNET_LIN_Database.

- **LIN2@nixnet: XNET_LIN_Database**—Uses LIN interface 2 of an NI-XNET device and so on with the form *LINx*.

Refer to the *NI-XNET Hardware and Software Manual* to assign a database cluster alias.

`baudrate`

The diagnostic communication baud rate. Default is –1, which reuses the baudrate of the selected LIN cluster from the assigned FIBEX or LDF database.

`MasterReqFrame`

Selects the Master Request Frame from an LDF or FIBEX database. If you assign an empty string (default) as `MasterReqFrame`, the name as defined in the LIN MasterReq standard is used.

`NAD`

NAD is the address of the slave node being addressed in a request. NAD also is used to indicate the source of a response. NAD values are 1–127 (0x7F), while 0 (zero) and 128 (0x80)–255 (0xFF) are reserved for other purposes.

`SlaveRespFrame`

Selects the Slave Response Frame from an LDF or FIBEX database. If you assign an empty string (default) as `SlaveRespFrame`, the name as defined in the LIN SlaveResp standard is used.

Output

`diagRefOut`

A struct that contains all necessary information about the diagnostic session. Pass this struct as a handle to all subsequent diagnostic functions and close it using `ndCloseDiagnostic`.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndOpenDiagnosticOnLIN` opens a diagnostic communication channel to an ECU. This function initializes the LIN port specified as input and stores a handle to it (among other internal data) into `diagRefOut`, which serves as reference for further diagnostic functions.

No communication to the ECU takes place at this point. To open a diagnostic session on the ECU, call `ndStartDiagnosticSession` or `ndUDSDiagnosticSessionControl`.

ndSetProperty

Purpose

Sets a diagnostic global internal parameter.

Format

```
void ndSetProperty(
       unsigned short propertyID,
       unsigned long propertyValue);
```

Input

`propertyID`

Defines the parameter whose value is to be modified:

0 **Timeout Diag Command** is the timeout in milliseconds the master waits for the response to a diagnostic request message. The default is 1000 ms.

1 **Timeout FC (Bs)** is the timeout in milliseconds the master waits for a Flow Control frame after sending a First Frame or the last Consecutive Frame of a block. The default is 250 ms.

2 **Timeout CF (Cr)** is the timeout in milliseconds the master waits for a Consecutive Frame in a multiframe response. The default is 250 ms for CAN and 1000 ms for LIN.

3 **Receive Block Size (BS)** is the number of Consecutive Frames the slave sends in one block before waiting for the next Flow Control frame. A value of 0 (default) means all Consecutive Frames are sent in one run without interruption.

4 **Wait Time CF (STmin)** defines the minimum time for the slave to wait between sending two Consecutive Frames of a block. Values from 0 to 127 are wait times in milliseconds. Values 241 to 249 (Hex F1 to F9) mean wait times of 100 µs to 900 µs, respectively. All other values are reserved. The default is 5 ms.

5 **Max Wait Frames (N_WFTmax)** is the maximum number of WAIT frames the master accepts before terminating the connection. The default is 10.

6 **Wait Frames to Send (N_WAIT)** is the number of WAIT frames the master sends every time before a CTS frame is sent. If you set this value to a negative number (for example, 0xFFFFFFFF = –1), the master sends an OVERLOAD frame instead of a WAIT, and reception is aborted. The default is 0 for maximum speed.

7 **Time between Waits (T_W)** is the number of milliseconds the master waits after sending a WAIT frame. The default is 25.

8 **Fill CAN Frames** specifies whether a CAN frame is transmitted with 8 bytes or less.

0: Short CAN frames are sent with DLC < 8.

1: Short CAN frames are filled to 8 bytes with **Fill Byte** (default).

9 **Fill Byte** specifies the CAN frame content, filled with defined data or random data.

0–255: Byte is used optionally to fill short CAN frames.

256: Short CAN frames are filled optionally with random bytes.

The default is 255 (0xFF).

10 **Invalid Response as Error** specifies how the toolkit handles an invalid ECU response.

0: Invalid response is indicated by `success` = FALSE only (default).

1: Invalid response is returned as an error in addition.

11 **Max RspPending Count** defines the number of times a ReqCorrectlyRcvd-RspPending (0x78) Negative Response Message will be accepted to extend the command timeout (default 5). If this message is sent more often in response to a request, an error –8120 is returned. If the ECU implements commands with a long duration (for example, flash commands), you may need to extend this number.

12 **VWTP Command Time Out** is the time in milliseconds the host waits for a VWTP 2.0 command to be executed (default 50 ms). The specification states this as 50 ms plus the network latency, but some ECUs may require higher values.

13 **STmin** sets the minimum time in seconds between the end of transmission of a frame in a diagnostic request message and the start of transmission of the next frame in the diagnostic request message for LIN-based diagnostic communication. The default is 0.

14 **P2min** sets the minimum time in seconds between reception of the last frame of the diagnostic request and the response sent by the node for LIN-based diagnostic communication. The default is 0.05.

15 **Termination** sets the NI-XNET Termination property. Reflections on the CAN and LIN bus can cause communication failures. To prevent reflections, termination can be present as external resistance or resistance the XNET CAN or LIN board applies internally. This property determines whether the XNET board uses termination to the bus. For further information about appropriate terminations of a CAN or LIN network, refer to the *NI-XNET Hardware and Software Manual*. The default is 0.

`propertyValue`

The requested property value.

Output

None.

Description

Use this function to set several internal diagnostic parameters, such as timeouts for the transport protocol. Use `ndGetProperty` to read them out.

ndStatusToString

Purpose

Returns a description for an error code.

Format

```
void ndStatusToString(
       long errorCode,
       char message[],
       long *len);
```

Input

`errorCode`

The status code (return value) of any other diagnostic functions.

Output

`message`

Returns a descriptive string for the error code.

`len`

On input, `len` must contain the `message` array length. On return, it contains the number of valid data bytes in the `message` array.

Description

When the status code returned from an Automotive Diagnostic Command Set function is nonzero, an error or warning is indicated. This function obtains an error/warning description for debugging purposes.

The return code is passed into the `errorCode` parameter. The `len` parameter indicates the number of bytes available in the string for the description. The description is truncated to size `len` if needed, but a size of 1024 characters is large enough to hold any description. The text returned in `message` is null-terminated, so you can use it with ANSI C functions such as `printf`. For C or C++ applications, each Automotive Diagnostic Command Set function returns a status code as a signed 32-bit integer. The following table summarizes the Automotive Diagnostic Command Set use of this status.

Status Code Use

Status Code	Definition
Negative	Error—Function did not perform the expected behavior.
Positive	Warning—Function performed as expected, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every Automotive Diagnostic Command Set function. If an error is detected, close all Automotive Diagnostic Command Set handles and exit the application. If a warning is detected, you can display a message for debugging purposes or simply ignore the warning.

The following code shows an example of handling Automotive Diagnostic Command Set status during application debugging.

```
Status = ndOpenDiagnostic ("CAN0", 500000, 0, 0x7E0, 0x7E8,
&MyDiagHandle);
PrintStat (status, "ndOpenDiagnostic");
```

where the function `PrintStat` has been defined at the top of the program as:

```
void PrintStat(mcTypeStatus status, char *source)
{
    char statusString[1024];
    long len = sizeof(statusString);
    if (status != 0)
    {
        ndStatusToString(status, statusString, &len);
        printf("\n%s\nSource = %s\n", statusString, source);
        if (status < 0)
        {
            ndCloseDiagnostic(&MyDiagHandle);
            exit(1);
        }
    }
}
```

ndVWTPConnect

Purpose

Establishes a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPConnect(
       TD1 *diagRef,
       unsigned long channelID,
       unsigned char applicationType);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`channelID`

Defines the CAN identifier on which the ECU responds for this connection. The ECU defines the ID on which the host transmits.

`applicationType`

Specifies the communication type that takes place on the communication channel. For diagnostic applications, specify *KWP2000 (1)*. The other values are for manufacturer-specific purposes.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must establish a connection to the ECU before any diagnostic communication can occur. This function sets up a unique communication channel to an ECU that you can use in subsequent diagnostic service requests.

You must maintain the communication link thus created by periodically (at least once a second) calling `ndVWTPConnectionTest`.

No equivalent exists for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

ndVWTPConnectionTest

Purpose

Maintains a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPConnectionTest(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must periodically maintain the connection link to the ECU, so that the ECU does not terminate it. You must execute this periodic refresh at least once per second.

This function sends a Connection Test message to the ECU and evaluates its response, performing the necessary steps to maintain the connection.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

ndVWTPDisconnect

Purpose

Terminates a connection channel to an ECU using the VW TP 2.0.

Format

```
long ndVWTPDisconnect(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

For the VW TP 2.0, you must disconnect the ECU connection link to properly terminate communication to the ECU. This function sends the proper disconnect messages and unlinks the communication.

Use `ndVWTPConnect` the create a new connection to the same ECU.

There is no equivalent for the ISO TP (ISO 15765-2), as the ISO TP does not use a special communication link.

DoIP Functions

ndDoIPActivateRouting

Purpose

Defines the source and target address for a DoIP TCP/IP connection.

Format

```
long ndDoIPActivateRouting (
       TD1 *diagRef,
       unsigned char ActivationType,
       unsigned short SourceAddress,
       unsigned short *TargetAddress);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ActivationType`

Indicates the specific type of routing activation that may require different types of authentication and/or confirmation. Defined values are:

0 Default.

1 WWH-OBD (worldwide harmonized onboard diagnostic).

0xE0 Use OEM-specific central security approach.

Values 2 to 0xDF are reserved. Values 0xE0 to 0xFF are OEM specific.

`SourceAddress`

The DoIP source address of the tester that starts the communication.

Output

`TargetAddress`

The logical address of the responding DoIP entity.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPActivateRouting` establishes a route for the DoIP messages and assigns an endpoint `TargetAddress`. After successfully establishing a route, diagnostic messages can be exchanged with the target DoIP entity using any of the diagnostic service functions.

ndDoIPConnect

Purpose

Creates a TCP/IP connection to a DoIP entity identified by its IP address.

Format

```
long ndDoIPConnect(
       TD1 *diagRef,
       char address[],
       unsigned short SourceAddress,
       unsigned short TargetAddress);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

The IP address of the DoIP entity to connect to (zero-terminated string in *a.b.c.d* notation).

`SourceAddress`

The DoIP source address of the tester that starts the communication. You can set this input to 0 if you are activating a route through `ndDoIPActivateRouting`.

`TargetAddress`

The DoIP target address of the device under test that should be connected to. You can set this input to 0 if you are activating a route through `ndDoIPActivateRouting`.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPConnect` creates a unique TCP/IP data connection to a certain DoIP entity identified by its IP address. The IP address might be retrieved from `ndDoIPGetEntities`. The TCP/IP data connection is needed to exchange diagnostic service requests.

You can specify `SourceAddress` and `TargetAddress` at this point or leave them blank if a routing activation is executed later using `ndDoIPActivateRouting`.

ndDoIPDisconnect

Purpose

Disconnects the TCP/IP connection to a DoIP entity.

Format

```
long ndDoIPDisconnect(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPDisconnect` terminates the TCP/IP connection to the connected DoIP entity. After executing this VI, diagnostic services no longer can be executed on that DoIP entity. You can reconnect with `ndDoIPConnect`.

ndDoIPEntityStatus

Purpose

Gets status information from a DoIP entity.

Format

```
long ndDoIPEntityStatus(
       TD1 *diagRef,
       unsigned char *nodeType,
       unsigned char *maxSockets,
       unsigned char *curSockets,
       LVBoolean *ok);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`nodeType`

Indicates the type of DoIP entity. Possible values are:

0 DoIP gateway

1 DoIP node

All other values are reserved.

`maxSockets`

Represents the maximum number of concurrent TCP/IP sockets allowed with this DoIP entity, excluding the reserve socket required for socket handling.

`curSockets`

The number of currently established TCP/IP sockets.

`ok`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPEntityStatus` serves the purpose of identifying certain operating conditions of the responding DoIP entity. For example, this allows for test equipment to detect existing diagnostic communication sessions as well as a DoIP entity's capabilities.

ndDoIPGetDiagPowerMode

Purpose

Gets information on the DoIP entity power state.

Format

```
long ndDoIPGetDiagPowerMode(
       TD1 *diagRef,
       unsigned char *powerMode,
       LVBoolean *ok);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`powerMode`

Identifies whether the vehicle is in Diagnostic Power Mode and ready to perform reliable diagnostics. Possible values are:

0 Not ready

1 Ready

All other values are reserved.

`ok`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPGetDiagPowerMode` retrieves a vehicle's Diagnostic Power Mode. For example, test equipment can use this information to verify whether the vehicle is in Diagnostic Power Mode, which allows for performing reliable diagnostics on the vehicle's components.

ndDoIPGetEntities

Purpose

Returns a table of all DoIP entities (vehicles) on the local subnet, possibly restricted to EID or VIN.

Format

```
long ndDoIPGetEntities(
       TD1 *diagRef,
       unsigned short DoIPOpenType,
       char VINOrEID[],
       unsigned char *DoIPEntities,
       unsigned long *len);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DoIPOpenType`

Defines which DoIP entities this command queries and lists. Allowed values are:

0 VIN (VIN is specified in `VINorEID`)

1 EID (EID is specified in `VINorEID`)

2 All (`VINorEID` is ignored)

Other values are reserved.

`VINOrEID`

Depends on `DoIPOpenType`:

`DoIPOpenType`	`VINOrEID` Value
0	A 17-character Vehicle Identification Number. Only DoIP entities for this VIN are listed.
1	An Entity ID (usually a MAC address). Only the DoIP entity with this ID is listed. Specify the EID as *xx-xx-xx-xx-xx-xx*, where each *x* stands for a hexadecimal digit.
2	Ignored.

Output

`DoIPEntities`

Returns an array of C structs, each of which describe a DoIP entity:

```
typedef struct
{
      char           VIN[18];
      unsigned short Address;
      unsigned char  EID[6];
      unsigned char  GID[6];
      char           IP_Address[16];
}
DOIP_ENTITY;
```

`VIN`	Contains the Vehicle Identification Number assigned to the DoIP entity. Could be empty if not assigned.
`Address`	The logical (Target) DoIP Address of the DoIP Entity.
`EID`	Contains the Entity ID (usually the Hardware MAC address) assigned to the DoIP entity. Could be empty (0) if not assigned.
`GID`	Contains the Group ID assigned to the DoIP entity. Could be empty(0) if not assigned.
`IP_Address`	Contains the DoIP Entity IP Address (in *a.b.c.d* notation).

`len`

Returns the size of the `DoIPEntities` array in bytes. Must be initialized with the size (in bytes) of the buffer provided for DoIPEntities.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPGetEntities` uses a UDP broadcast to identify all DoIP entities in the local subnet matching a certain condition. The entities responding are returned in the `DoIPEntities` cluster array.

The conditions are either a common VIN or EID or simply all entities connected. Refer to the description of `DoIPOpenType` and `VINOrEID`.

ndDoIPSendVehicleIdentRequest

Purpose

Sends a UDP request to all DoIP-capable vehicles in the local subnet to identify themselves.

Format

```
long ndDoIPSendVehicleIdentRequest(
       TD1 *diagRef);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPSendVehicleIdentRequest` sends a Vehicle Identification Request to all DoIP entities in the local subnet.

Usually, this is done as part of `ndDoIPGetEntities` and does not need to be executed separately.

ndDoIPSendVehicleIdentReqEID

Purpose

Sends a UDP request to all DoIP-capable vehicles with a certain EID (MAC address) in the local subnet to identify themselves.

Format

```
long ndDoIPSendVehicleIdentReqEID(
       TD1 *diagRef,
       char EID[]);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`EID`

The Entity ID (usually the MAC address) of the DoIP entity that is assumed to respond. Specify the EID as *xx-xx-xx-xx-xx-xx*, where each *x* stands for a hexadecimal digit (zero-terminated string).

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPSendVehicleIdentReqEID` sends a Vehicle Identification Request to all DoIP entities in the local subnet identified by the given `EID`.

Usually, this is done as part of `ndDoIPGetEntities` and does not need to be executed separately.

ndDoIPSendVehicleIdentReqVIN

Purpose

Sends a UDP request to all DoIP-capable vehicles with a certain VIN (Vehicle Identification Number) in the local subnet to identify themselves.

Format

```
long ndDoIPSendVehicleIdentReqVIN(
       TD1 *diagRef,
       char VIN[]);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`VIN`

The 17-character Vehicle Identification Number of the DoIP entity that is assumed to respond (zero-terminated string).

Output

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndDoIPSendVehicleIdentReqVIN` sends a Vehicle Identification Request to all DoIP entities in the local subnet identified by the given `VIN`.

Usually, this is done as part of `ndDoIPGetEntities` and does not need to be executed separately.

KWP2000 Services

ndClearDiagnosticInformation

Purpose

Executes the ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndClearDiagnosticInformation(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       TD3 *DTCDescriptor,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
    long DTCByteLength;
    long StatusByteLength;
    long AddDataByteLength;
    unsigned short ByteOrder;
    } TD3;
```

DTCByteLength indicates the number of bytes the ECU sends for each DTC. The default is 2.

StatusByteLength indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

AddDataByteLength indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

ByteOrder indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the DTCs struct array.

Output

success

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the ndStatusToString function to obtain a descriptive string for the return value.

Description

This function clears the diagnostic information on the ECU memory. groupOfDTC specifies the type of diagnostic trouble codes to be cleared on the ECU memory.

For further details about this service, refer to the ISO 14230-3 standard.

ndControlDTCSetting

Purpose

Executes the ControlDTCSetting service. Modifies the generation behavior of selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndControlDTCSetting(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       unsigned char dataIn[],
       long len,
       TD3 *DTCDescriptor,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manipulate the elements of this struct manually.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be controlled. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network-related DTCs
0xFF00	All DTCs

`dataIn`

Specifies application-specific data that control DTC generation.

`len`

Must contain the number of valid data bytes in `dataIn`.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
   long DTCByteLength;
   long StatusByteLength;
   long AddDataByteLength;
   unsigned short ByteOrder;
   } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. Default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. Default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

The response byte stream is interpreted according to this description, and the resulting DTC records are returned in the `DTCs` struct array.

For this service, `DTCByteLength` and `ByteOrder` are used to format the `groupOfDTC` parameter correctly into the request message.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndDisableNormalMessageTransmission

Purpose

Executes the DisableNormalMessageTransmission service. The ECU no longer transmits its regular communication messages (usually CAN messages).

Format

```
long ndDisableNormalMessageTransmission(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndECUReset

Purpose

Executes the ECUReset service. Resets the ECU.

Format

```
long ndECUReset(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the reset mode:

Hex	Description
01	**PowerOn** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the client (tester) re-establishes communication.
02	**PowerOnWhileMaintainingCommunication** This value identifies the PowerOn ResetMode, a simulated PowerOn reset that most ECUs perform after the ignition OFF/ON cycle. When the ECU performs the reset, the server (ECU) maintains communication with the client (tester).
03–7F	**Reserved**
80–FF	**ManufacturerSpecific** This range of values is reserved for vehicle manufacturer-specific use.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests the ECU to perform an ECU reset effectively based on the `mode` value content. The vehicle manufacturer determines when the positive response message is sent.

ndEnableNormalMessageTransmission

Purpose

Executes the EnableNormalMessageTransmission service. The ECU starts transmitting its regular communication messages (usually CAN messages).

Format

```
long ndEnableNormalMessageTransmission(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndInputOutputControlByLocalIdentifier

Purpose

Executes the InputOutputControlByLocalIdentifier service. Modifies the ECU I/O port behavior.

Format

```
long ndInputOutputControlByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the I/O to be manipulated. The values are application specific.

`mode`

Defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ReportCurrentState

4: ResetToDefault

5: FreezeCurrentState

7: ShortTermAdjustment

8: LongTermAdjustment

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by `localID`.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadDataByLocalIdentifier

Purpose

Executes the ReadDataByLocalIdentifier service. Reads an ECU data record.

Format

```
long ndReadDataByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the data to be read. The values are application specific.

Output

`dataOut`

Returns the data record from the ECU. If you know the record data description, you can use the `ndConvertToPhys` function to interpret it.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests data record values from the ECU identified by the `localID` parameter.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadDTCByStatus

Purpose

Executes the ReadDiagnosticTroubleCodesByStatus service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndReadDTCByStatus(
       TD1 *diagRef,
       unsigned char mode,
       unsigned short groupOfDTC,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Defines the type of DTCs to be read. The values are application specific. The usual values are:

2: AllIdentified

3: AllSupported

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
        unsigned long DTC;
        unsigned long Status;
        unsigned long AddData;
        } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear

6 testNotCompletedThisMonitoringCycle

7 warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function reads diagnostic trouble codes by status from the ECU memory. If you set the optional `groupOfDTC` parameter to the above specified codes, the ECU reports DTCs only with status information based on the functional group selected by `groupOfDTC`.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadECUIdentification

Purpose

Executes the ReadECUIdentification service. Returns ECU identification data.

Format

```
long ndReadECUIdentification(
       TD1 *diagRef,
       unsigned char mode,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the type of identification information to be returned. The values are application specific.

Output

`dataOut`

Returns the ECU identification data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests identification data from the ECU. `mode` identifies the type of identification data requested. The ECU returns identification data that `dataOut` can access. The `dataOut` format and definition are vehicle manufacturer specific.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadMemoryByAddress

Purpose

Executes the ReadMemoryByAddress service. Reads data from the ECU memory.

Format

```
long ndReadMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address from which data are read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be read.

Output

`dataOut`

Returns the ECU memory data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests ECU memory data identified by the `address` and `size` parameters. The `dataOut` format and definition are vehicle manufacturer specific. `dataOut` includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

For further details about this service, refer to the ISO 14230-3 standard.

ndReadStatusOfDTC

Purpose

Executes the ReadStatusOfDiagnosticTroubleCodes service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndReadStatusOfDTC(
       TD1 *diagRef,
       unsigned short groupOfDTC,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The following values have a special meaning:

0x0000	All powertrain DTCs
0x4000	All chassis DTCs
0x8000	All body DTCs
0xC000	All network related DTCs
0xFF00	All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

StatusByteLength indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

AddDataByteLength indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

ByteOrder indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the DTCs struct array.

Output

DTCs

Returns the resulting DTCs as an array of structs:

```
typedef struct {
        unsigned long DTC;
        unsigned long Status;
        unsigned long AddData;
        } TD4;
```

DTC is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use ndDTCToString to convert this code to readable format as defined by SAE J2012.

Status is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

AddData contains optional additional data for this DTC. Usually, this does not contain valid information (refer to DTCDescriptor).

len

On input, len must contain the DTCs array length in elements. On return, it contains the number of valid elements in the DTCs array.

success

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function reads diagnostic trouble codes from the ECU memory. If you specify `groupOfDTC`, the ECU reports DTCs based only on the functional group selected by `groupOfDTC`.

For further details about this service, refer to the ISO 14230-3 standard.

ndRequestRoutineResultsByLocalIdentifier

Purpose

Executes the RequestRoutineResultsByLocalIdentifier service. Returns results from an ECU routine.

Format

```
long ndRequestRoutineResultsByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine from which this function retrieves results. The values are application specific.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests results (for example, exit status information) referenced by `localID` and generated by the routine executed in the ECU memory.

For further details about this service, refer to the ISO 14230-3 standard.

ndRequestSeed

Purpose

Executes the SecurityAccess service to retrieve a seed from the ECU.

Format

```
long ndRequestSeed(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char seedOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

Output

`seedOut`

Returns the seed from the ECU.

`len`

On input, `len` must contain the `seedOut` array length. On return, it contains the number of valid data bytes in the `seedOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndSendKey

Purpose

Executes the SecurityAccess service to send a key to the ECU.

Format

```
long ndSendKey(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char keyIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

`keyIn`

Defines the key data to be sent to the ECU.

`len`

Must contain the number of valid data bytes in `keyIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndStartDiagnosticSession

Purpose

Executes the StartDiagnosticSession service. The ECU is set up in a specific diagnostic mode.

Format

```
long ndStartDiagnosticSession(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the diagnostic mode into which the ECU is brought. The values are application specific.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function enables different ECU diagnostic modes. The possible diagnostic modes are not defined in ISO 14230 and are application specific. A diagnostic session starts only if communication with the ECU is established. For more details about starting communication, refer to ISO 14230-2. If no diagnostic session is requested after `ndOpenDiagnostic`, a default session is enabled automatically in the ECU. The default session supports at least the following services:

- The StopCommunication service (refer to `ndCloseDiagnostic` and the ISO 14230-2 standard).
- The TesterPresent service (refer to `ndTesterPresent` and the ISO 14230-3 standard).

ndStartRoutineByLocalIdentifier

Purpose

Executes the StartRoutineByLocalIdentifier service. Executes a routine on the ECU.

Format

```
long ndStartRoutineByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine to be started. The values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function starts a routine in the ECU memory. The ECU routine starts after the positive response message is sent. The routine stops until the `ndStopRoutineByLocalIdentifier` function and corresponding service are issued. The routines could be either tests that run instead of normal operating code or routines enabled and executed with the normal operating code running. In the first case, you may need to switch the ECU to a specific diagnostic mode using `ndOpenDiagnostic` or unlock the ECU using the SecurityAccess service prior to using `ndStartRoutineByLocalIdentifier`.

For further details about this service, refer to the ISO 14230-3 standard.

ndStopDiagnosticSession

Purpose

Executes the StopDiagnosticSession service. Returns the ECU to normal mode.

Format

```
long ndStopDiagnosticSession(
       TD1 *diagRef,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function disables the current ECU diagnostic mode. A diagnostic session stops only if communication with the ECU is established and a diagnostic session is running. If no diagnostic session is running, the default session is active. `ndStopDiagnosticSession` cannot disable the default session. If the ECU stops the current diagnostic session, it performs the necessary action to restore its normal operating conditions. Restoring the normal ECU operating conditions may include resetting all controlled actuators activated during the diagnostic session being stopped, and resuming all normal ECU algorithms. You should call `ndStopDiagnosticSession` before disabling communication with `ndCloseDiagnostic`, but only if you previously used `ndStartDiagnosticSession`.

ndStopRoutineByLocalIdentifier

Purpose

Executes the StopRoutineByLocalIdentifier service. Stops a routine on the ECU.

Format

```
long ndStopRoutineByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the routine to be stopped. The values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function stops a routine in the ECU memory referenced by `localID`.

For further details about this service, refer to the ISO 14230-3 standard.

ndTesterPresent

Purpose

Executes the TesterPresent service. Keeps the ECU in diagnostic mode.

Format

```
long ndTesterPresent(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

Keep calling `ndTesterPresent` within the ECU timeout period if no other service is executed.

ndWriteDataByLocalIdentifier

Purpose

Executes the WriteDataByLocalIdentifier service. Writes a data record to the ECU.

Format

```
long ndWriteDataByLocalIdentifier(
       TD1 *diagRef,
       unsigned char localID,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`localID`

Defines the local identifier of the data to be read. The values are application specific.

`dataIn`

Defines the data record to be written to the ECU. If you know the record data description, use `ndConvertFromPhys` to generate this record.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the WriteDataByLocalIdentifier service and writes RecordValues (data values) to the ECU. `dataIn` identifies the data values to be transmitted. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

ndWriteMemoryByAddress

Purpose

Executes the WriteMemoryByAddress service. Writes data to the ECU memory.

Format

```
long ndWriteMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address to which data are written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be written.

`dataIn`

Defines the memory block to be written to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This VI performs the KWP2000 WriteDataByAddress service and writes RecordValues (data values) to the ECU. `address` and `size` identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 14230-3 standard.

UDS (DiagOnCAN) Services

ndUDSClearDiagnosticInformation

Purpose

Executes the UDS ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSClearDiagnosticInformation(
       TD1 *diagRef,
       unsigned long groupOfDTC,
       TD3 *DTCDescriptor,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`groupOfDTC`

Specifies the group of diagnostic trouble codes to be cleared. The values are application specific. The following value has a special meaning:

0xFFFFFF All DTCs

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
   long DTCByteLength;
   long StatusByteLength;
   long AddDataByteLength;
   unsigned short ByteOrder;
   } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function clears the diagnostic information on the ECU memory. Depending on the value of `groupOfDTC`, the ECU is requested to clear the corresponding DTCs. The `groupOfDTC` values are application specific.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSCommunicationControl

Purpose

Executes the UDS CommunicationControl service. Switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off.

Format

```
long ndUDSCommunicationControl(
       TD1 *diagRef,
       unsigned char type,
       unsigned char communicationType,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`type`

Indicates whether transmission/reception is to be switched on/off. The usual values are:

00: enableRxAndTx

01: enableRxAndDisableTx

02: disableRxAndEnableTx

03: disableRxAndTx

`communicationType`

A bitfield indicating which application level is to be changed. The usual values are:

01: application

02: networkManagement

You can change more than one level at a time.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the UDS CommunicationControl service and switches transmission and/or reception of the normal communication messages (usually CAN messages) on or off. The `type` and `communication type` parameters are vehicle manufacturer specific (one OEM may disable the transmission only, while another OEM may disable the transmission and reception based on vehicle manufacturer specific needs). The request is either transmitted functionally addressed to all ECUs with a single request message, or transmitted physically addressed to each ECU in a separate request message.

ndUDSControlDTCSetting

Purpose

Executes the UDS ControlDTCSetting service. Modifies Diagnostic Trouble Code (DTC) behavior.

Format

```
long ndUDSControlDTCSetting(
       TD1 *diagRef,
       unsigned char type,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`type`

Specifies the control mode:

1: on

2: off

`dataIn`

Specifies application-specific data that control DTC generation.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndUDSDiagnosticSessionControl

Purpose

Executes the UDS DiagnosticSessionControl service. The ECU is set up in a specific diagnostic mode.

Format

```
long ndUDSDiagnosticSessionControl(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the diagnostic mode into which the ECU is brought. The values are application specific. The usual values are:

01: defaultSession

02: ECUProgrammingSession

03: ECUExtendedDiagnosticSession

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndUDSECUReset

Purpose

Executes the UDS ECUReset service. Resets the ECU.

Format

```
long ndUDSECUReset(
       TD1 *diagRef,
       unsigned char mode,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Indicates the reset mode:

Hex	Description
01	hardReset
02	keyOffOnReset
03	softReset
04	enableRapidPowerShutDown
05	disableRapidPowerShutDown

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests the ECU to perform an ECU reset effectively based on the `mode` parameter value content. The vehicle manufacturer determines when the positive response message is sent. Depending the value of `mode`, the corresponding ECU reset event is executed as a hard reset, key off/on reset, soft reset, or other reset.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSInputOutputControlByIdentifier

Purpose

Executes the UDS InputOutputControlByIdentifier service. Modifies ECU I/O port behavior.

Format

```
long ndUDSInputOutputControlByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the I/O to be manipulated. The values are application specific.

`mode`

Defines the I/O control type. The values are application specific. The usual values are:

0: ReturnControlToECU

1: ResetToDefault

2: FreezeCurrentState

3: ShortTermAdjustment

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function substitutes a value for an input signal or internal ECU function. It also controls an output (actuator) of an electronic system referenced by the `ID` parameter.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReadDataByIdentifier

Purpose

Executes the UDS ReadDataByIdentifier service. Reads an ECU data record.

Format

```
long ndUDSReadDataByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the data to be read. The values are application specific.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests data record values from the ECU identified by the `ID` parameter.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReadMemoryByAddress

Purpose

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format

```
long ndUDSReadMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address from which data are read. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be read.

Output

`dataOut`

Returns the ECU memory data.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function requests memory data from the ECU identified by the `address` and `size` parameters. The `dataOut` format and definition are vehicle manufacturer specific. `dataOut` includes analog input and output signals, digital input and output signals, internal data, and system status information if the ECU supports them.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportDTCBySeverityMaskRecord

Purpose

Executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportDTCBySeverityMaskRecord(
       TD1 *diagRef,
       unsigned char severityMask,
       unsigned char status,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`severityMask`

Defines the status of DTCs to be read. The values are application specific.

`status`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC.

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportDTCBySeverityMaskRecord subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportDTCByStatusMask

Purpose

Executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportDTCByStatusMask(
       TD1 *diagRef,
       unsigned char statusMask,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`statusMask`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportDTCByStatusMask subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportSeverityInformationOfDTC

Purpose

Executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads selected Diagnostic Trouble Codes (DTCs) are read.

Format

```
long ndUDSReportSeverityInformationOfDTC(
       TD1 *diagRef,
       unsigned long DTCMaskRecord,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCMaskRecord`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. For this subfunction, the default is 2.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC.

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportSeverityInformationOfDTC subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads the selected DTCs from the ECU memory.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSReportSupportedDTCs

Purpose

Executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service. Reads all supported Diagnostic Trouble Codes (DTCs).

Format

```
long ndUDSReportSupportedDTCs(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       unsigned char *statusAvailMask,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3 for UDS.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`statusAvailMask`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the ReportSupportedDTCs subfunction of the UDS ReadDiagnosticTroubleCodeInformation service and reads all supported DTCs from the ECU memory.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSRequestDownload

Purpose

Initiates a download of data to the ECU.

Format

```
long ndUDSRequestDownload (
       TD1 *diagRef,
       unsigned long memoryAddress,
       unsigned long memorySize,
       unsigned char memoryAddressLength,
       unsigned char memorySizeLength,
       unsigned char dataFormatIdentifier,
       unsigned long *blockSize,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memoryAddress`

Defines the memory address to which data are to be written.

`memorySize`

Defines the size of the data to be written.

`memoryAddressLength`

Defines the number of bytes of the `memoryAddress` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

`memorySizeLength`

Defines the number of bytes of the `memorySize` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

`dataFormatIdentifier`

Defines the compression and encryption scheme for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

Output

`blockSize`

Returns the number of data bytes to be transferred to the ECU in subsequent `ndUDSTransferData` requests.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSRequestDownload` initiates the download of a data block to the ECU. This is required to set up the download process; the actual data transfer occurs with subsequent `ndUDSTransferData` requests. The transfer must occur in blocks of the size this service returns (the `blockSize` parameter). After the download completes, use the `ndUDSRequestTransferExit` service to terminate the process.

ndUDSRequestSeed

Purpose

Executes the UDS SecurityAccess service to retrieve a seed from the ECU.

Format

```
long ndUDSRequestSeed(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char seedOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an odd number, usually 1.

Output

`seedOut`

Returns the seed from the ECU.

`len`

On input, `len` must contain the `seedOut` array length. On return, it contains the number of valid data bytes in the `seedOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndUDSRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndUDSSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndUDSRequestTransferExit

Purpose

Terminates a download/upload process.

Format

```
long ndUDSRequestTransferExit (
       TD1 *diagRef,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`dataIn`

Defines a data record to be written to the ECU as part of the termination process. The meaning is implementation dependent; this may be a checksum or a similar verification instrument.

`len`

Must be set to the buffer size for the `dataIn` parameter.

Output

`dataOut`

Returns a memory data block from the ECU as part of the termination process. The meaning is implementation dependent; this may be a checksum or a similar verification instrument.

`len2`

Must be set to the buffer size for the `dataOut` parameter. On return, it contains the actual data size returned in `dataOut`.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the ndStatusToString function to obtain a descriptive string for the return value.

Description

ndUDSRequestTransferExit terminates a download or upload process initialized with ndUDSRequestDownload or ndUDSRequestUpload.

ndUDSRequestUpload

Purpose

Initiates an upload of data from the ECU.

Format

```
long ndUDSRequestUpload (
       TD1 *diagRef,
       unsigned long memoryAddress,
       unsigned long memorySize,
       unsigned char memoryAddressLength,
       unsigned char memorySizeLength,
       unsigned char dataFormatIdentifier,
       unsigned long *blockSize,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memoryAddress`

Defines the memory address from which data are to be read.

`memorySize`

Defines the size of the data to be read.

`memoryAddressLength`

Defines the number of bytes of the `memoryAddress` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the address are written to the ECU.

`memorySizeLength`

Defines the number of bytes of the `memorySize` parameter that are written to the ECU. This value is implementation dependent and must be in the range of 1–4. For example, if this value is 2, only the two lowest bytes of the size are written to the ECU.

`dataFormatIdentifier`

Defines the compression and encryption scheme used for the data blocks written to the ECU. A value of 0 means no compression/no encryption. Nonzero values are not standardized and implementation dependent.

Output

`blockSize`

Returns the number of data bytes to be transferred from the ECU in subsequent `ndUDSTransferData` requests.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSRequestUpload` initiates the upload of a data block from the ECU. This is required to set up the upload process; the actual data transfer occurs with subsequent `ndUDSTransferData` requests. The transfer must occur in blocks of the size that this service returns (the `blockSize` parameter). After the download completes, use the `ndUDSRequestTransferExit` service to terminate the process.

ndUDSRoutineControl

Purpose

Executes the UDS RoutineControl service. Executes a routine on the ECU.

Format

```
long ndUDSRoutineControl(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char mode,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the routine to be started. The values are application specific.

`mode`

Defines the operation mode for this service:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

`dataIn`

Defines application-specific input parameters for the routine.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function executes the UDS RoutineControl service and launches an ECU routine, stops an ECU routine, or requests ECU routine results from the ECU.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSSendKey

Purpose

Executes the UDS SecurityAccess service to send a key to the ECU.

Format

```
long ndUDSSendKey(
       TD1 *diagRef,
       unsigned char accessMode,
       unsigned char keyIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`accessMode`

Indicates the security level to be granted. The values are application specific. This is an even number, usually 2.

`keyIn`

Defines the key data to be sent to the ECU.

`len`

Must contain the number of valid data bytes in `keyIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The usual procedure for getting a security access to the ECU is as follows:

1. Request a seed from the ECU using `ndUDSRequestSeed` with access mode = *n*.
2. From the seed, compute a key for the ECU on the host.
3. Send the key to the ECU using `ndUDSSendKey` with access mode = *n* + 1.
4. The security access is granted if the ECU validates the key sent. Otherwise, an error is returned.

ndUDSTesterPresent

Purpose

Executes the UDS TesterPresent service. Keeps the ECU in diagnostic mode.

Format

```
long ndUDSTesterPresent(
       TD1 *diagRef,
       LVBoolean *requireResponse,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`requireResponse`

Indicates whether a response to this service is required. If `*requireResponse` is FALSE, no response is evaluated, and `success` is always returned TRUE. This parameter is passed by reference.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

To ensure proper ECU operation, you may need to keep the ECU informed that a diagnostic session is still in progress. If you do not send this information (for example, because the communication is broken), the ECU returns to normal mode from diagnostic mode after a while.

The TesterPresent service is this “keep alive” signal. It does not affect any other ECU operation.

Keep calling `ndUDSTesterPresent` within the ECU timeout period if no other service is executed.

ndUDSTransferData

Purpose

Transfers data to/from the ECU in a download/upload process.

Format

```
long ndUDSTransferData (
       TD1 *diagRef,
       unsigned char *blockSequenceCounter,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`blockSequenceCounter`

Used to number the data blocks to be transferred to/from the ECU. The block sequence counter value starts at 01 hex with the first `ndUDSTransferData` request that follows the `ndUDSRequestDownload` or `ndUDSRequestUpload` service. Its value is incremented by 1 for each subsequent `ndUDSTransferData` request. At the value of FF hex, the block sequence counter rolls over and starts at 00 hex with the next `ndUDSTransferData` request.

The block sequence counter is updated automatically, and the updated value is returned.

`dataIn`

Defines the data block to be written to the ECU.

For a download, this is a memory data block to be downloaded to the ECU.

For an upload, the meaning is implementation dependent.

`len`

Must be set to the buffer size for the `dataIn` parameter.

Output

`blockSequenceCounter`

Returns the updated value of the block sequence counter (refer to the description in the *Input* section).

`dataOut`

Returns the memory data from the ECU.

For a download, this may contain a checksum or similar verification instrument; the meaning is implementation dependent.

For an upload, this is a memory data block uploaded from the ECU.

`len2`

Must be set to the buffer size for the `dataOut` parameter. On return, it contains the actual data size returned in `dataOut`.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

`ndUDSTransferData` executes the data transfer of a download process (initiated with a previous `ndUDSRequestDownload` request) or an upload process (initiated with a previous `ndUDSRequestUpload` request). The data transfer must occur in blocks of the size that has been returned in the block size parameter of the respective request service. After the data transfer completes, terminate the operation by calling the `ndUDSRequestTransferExit` service.

ndUDSWriteDataByIdentifier

Purpose

Executes the UDS WriteDataByIdentifier service. Writes a data record to the ECU.

Format

```
long ndUDSWriteDataByIdentifier(
       TD1 *diagRef,
       unsigned short ID,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`ID`

Defines the identifier of the data to be read. The values are application specific.

`dataIn`

Defines the data record written to the ECU. If you know the record data description, use `ndConvertFromPhys` to generate this record.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the UDS service WriteDataByIdentifier and writes RecordValues (data values) into the ECU. `dataIn` identifies the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDSWriteMemoryByAddress

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format

```
long ndUDSWriteMemoryByAddress(
       TD1 *diagRef,
       unsigned long address,
       unsigned char size,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`address`

Defines the memory address to which data are written. Only three bytes are sent to the ECU, so the address must be in the range 0–FFFFFF (hex).

`size`

Defines the length of the memory block to be written.

`dataIn`

Defines the memory block to be written to the ECU.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

This function performs the UDS service WriteMemoryByAddress and writes RecordValues (data values) into the ECU. `address` and `size` identify the data. The vehicle manufacturer must ensure the ECU conditions are met when performing this service. Typical use cases are clearing nonvolatile memory, resetting learned values, setting option content, setting the Vehicle Identification Number, or changing calibration values.

For further details about this service, refer to the ISO 15765-3 standard.

ndUDS06ReadMemoryByAddress

Executes the UDS ReadMemoryByAddress service. Reads data from the ECU memory.

Format

```
long ndUDS06ReadMemoryByAddress(
       TD1 *diagRef,
       unsigned char memAddrLen,
       unsigned char memSizeLen,
       unsigned long address,
       unsigned long size,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memAddrLen`

Defines the number of `address` bytes transferred to the ECU. This implicitly defines the maximum allowed for the `address` parameter.

`memSizeLen`

Defines the number of `size` bytes transferred to the ECU. This implicitly defines the maximum allowed for the `size` parameter.

`address`

Defines the memory address from which data are read.

`size`

Defines the length of the memory block to be read.

Output

`dataOut`

Returns the memory data from the ECU.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

Similar to `ndUDSReadMemoryByAddress`. You can define the address and size parameter sizes in bytes.

ndUDS06WriteMemoryByAddress

Purpose

Executes the UDS WriteMemoryByAddress service. Writes data to the ECU memory.

Format

```
long ndUDS06WriteMemoryByAddress(
       TD1 *diagRef,
       unsigned char memAddrLen,
       unsigned char memSizeLen,
       unsigned long address,
       unsigned long size,
       unsigned char dataIn[],
       long len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the handle for the diagnostic session, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`memAddrLen`

Defines the number of `address` bytes transferred to the ECU. This implicitly defines the maximum allowed for the `address` parameter.

`memSizeLen`

Defines the number of `size` bytes transferred to the ECU. This implicitly defines the maximum allowed for the `size` parameter.

`address`

Defines the memory address to which data are written.

`size`

Defines the length of the memory block to be written.

`dataIn`

Defines the memory block to be written to the ECU.

`len`

Must contain the number of (valid) data bytes in dataIn.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

Similar to the `ndUDSWriteMemoryByAddress`. You can define the `address` and `size` parameter sizes in bytes.

OBD (On-Board Diagnostics) Services

ndOBDClearEmissionRelatedDiagnosticInformation

Purpose

Executes the OBD Clear Emission Related Diagnostic Information service. Clears emission-related diagnostic trouble codes (DTCs) in the ECU.

Format

```
long ndOBDClearEmissionRelatedDiagnosticInformation(
       TD1 *diagRef,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestControlOfOnBoardDevice

Purpose

Executes the OBD Request Control Of On-Board Device service. Modifies ECU I/O port behavior.

Format

```
long ndOBDRequestControlOfOnBoardDevice(
       TD1 *diagRef,
       unsigned char TID,
       unsigned char dataIn[],
       long len,
       unsigned char dataOut[],
       long *len2,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`TID`

Defines the test identifier of the I/O to be manipulated. The values are application specific.

`dataIn`

Defines application-specific data for this service.

`len`

Must contain the number of valid data bytes in `dataIn`.

Output

`dataOut`

Returns application-specific data for this service.

`len2`

On input, `len2` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestCurrentPowertrainDiagnosticData

Purpose

Executes the OBD Request Current Powertrain Diagnostic Data service. Reads an ECU data record.

Format

```
long ndOBDRequestCurrentPowertrainDiagnosticData(
       TD1 *diagRef,
       unsigned char PID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`PID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestEmissionRelatedDTCs

Purpose

Executes the OBD Request Emission Related DTCs service. Reads all emission-related Diagnostic Trouble Codes (DTCs).

Format

```
long ndOBDRequestEmissionRelatedDTCs(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle

Purpose

Executes the OBD Request Emission Related DTCs During Current Drive Cycle service. Reads the emission-related Diagnostic Trouble Codes (DTCs) that occurred during the current (or last completed) drive cycle.

Format

```
long ndOBDRequestEmissionRelatedDTCsDuringCurrentDriveCycle(
        TD1 *diagRef,
        TD3 *DTCDescriptor,
        TD4 DTCs[],
        long *len,
        LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
        long DTCByteLength;
        long StatusByteLength;
        long AddDataByteLength;
        unsigned short ByteOrder;
        } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestOnBoardMonitoringTestResults

Purpose

Executes the OBD Request On-Board Monitoring Test Results service. Reads an ECU test data record.

Format

```
long ndOBDRequestOnBoardMonitoringTestResults(
       TD1 *diagRef,
       unsigned char OBDMID,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`OBDMID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

Output

`dataOut`

Returns the ECU test data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestPermanentFaultCodes

Purpose

Executes the OBD Request Permanent Fault Codes service. All permanent Diagnostic Trouble Codes (DTCs) are read.

Format

```
long ndOBDRequestPermanentFaultCodes(
       TD1 *diagRef,
       TD3 *DTCDescriptor,
       TD4 DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
       long DTCByteLength;
       long StatusByteLength;
       long AddDataByteLength;
       unsigned short ByteOrder;
       } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 2.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 0 for OBD.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. For the default 2-byte DTCs, use `ndDTCToString` to convert this code to readable format as defined by SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisMonitoringCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear
6	testNotCompletedThisMonitoringCycle
7	warningIndicatorRequested

For OBD, this field usually does not contain valid information.

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the DTCs array length in elements. On return, it contains the number of valid elements in the DTCs array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestPowertrainFreezeFrameData

Purpose

Executes the OBD Request Powertrain Freeze Frame Data service. Reads an ECU data record stored while a diagnostic trouble code occurred.

Format

```
long ndOBDRequestPowertrainFreezeFrameData(
       TD1 *diagRef,
       unsigned char PID,
       unsigned char nFrame,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`PID`

Defines the parameter identifier of the data to be read. The SAE J1979 standard defines the values.

`nFrame`

The number of the freeze frame from which the data are to be retrieved.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, use `ndConvertToPhys` to interpret this record. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndOBDRequestVehicleInformation

Purpose

Executes the OBD Request Vehicle Information service. Reads a set of information data from the ECU.

Format

```
long ndOBDRequestVehicleInformation(
       TD1 *diagRef,
       unsigned char infoType,
       unsigned char *nItems,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` or `ndOpenDiagnosticOnIP` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`infoType`

Defines the type of information to be read. The SAE J1979 standard defines the values.

Output

`nItems`

The number of data items (not bytes) this service returns.

`dataOut`

Returns the ECU vehicle information. You can obtain the description from the SAE J1979 standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) Services

ndWWHOBDClearEmissionRelatedDTCs

Purpose

Executes the WWH-OBD ClearDiagnosticInformation service. Clears selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndWWHOBDClearEmissionRelatedDTCs (
       TD1 *diagRef,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD ClearDiagnosticInformation service is based on the UDS ClearDiagnosticInformation service (ISO 14229-1).

ndWWHOBDConvertDTCsToJ1939

Purpose

Converts DTCs to the J1939 DTC format.

Format

```
long ndWWHOBDConvertDTCsToJ1939 (
       TD4 DTCs[],
       long lenDTCs,
       TD5 DTCsJ1939[],
       long *lenDTCsJ1939);
```

Input

`DTCs`

The DTC to convert as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`lenDTCs`

Must contain the number of valid data bytes in DTCs.

Output

`DTCsJ1939`

Returns the converted DTCs to the J1939 format as an array of structs:

```
typedef struct {
   unsigned long SPN;
   unsigned long Status;
   unsigned long AddData;
   unsigned long FMI;
   } TD5;
```

`SPN` contains the suspect parameter number for this DTC.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisOperationCycle
2	pendingDTC
3	confirmedDTC

4 testNotCompletedSinceLastClear

5 testFailedSinceLastClear

6 testNotCompletedThisOperationCycle

7 warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`FMI` contains the failure mode identifier.

`lenDTCsJ1939`

On input, `lenDTCsJ1939` must contain the `DTCsJ1939` array length. On return, it contains the number of valid data bytes in the `DTCsJ1939` array.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndWWHOBDConvertDTCsToJ2012

Purpose

Converts DTCs to the J2012 DTC format.

Format

```
long ndWWHOBDConvertDTCsToJ2012 (
       TD4 DTCs[],
       long lenDTCs,
       TD5 DTCsJ2012[],
       long *lenDTCsJ2012);
```

Input

`DTCs`

The DTC to convert as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`lenDTCs`

Must contain the number of valid data bytes in `DTCs`.

Output

`DTCsJ2012`

Returns the converted DTCs to the J2012 format as an array of structs:

```
typedef struct {
   unsigned short DTC;
   unsigned char FTB;
   unsigned long Status;
   unsigned long AddData;
   } TD6;
```

`DTC` is the resulting Diagnostic Trouble Code.

`FTB` contains the failure type byte.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisOperationCycle
2	pendingDTC

3 confirmedDTC

4 testNotCompletedSinceLastClear

5 testFailedSinceLastClear

6 testNotCompletedThisOperationCycle

7 warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`lenDTCsJ2012`

On input, `lenDTCsJ2012` must contain the `DTCsJ2012` array length. On return, it contains the number of valid data bytes in the `DTCsJ2012` array.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

ndWWHOBDRequestDID

Purpose

Executes the WWH-OBD ReadDataByIdentifier service. Reads a data record from the ECU.

Format

```
long ndWWHOBDRequestDID (
       TD1 *diagRef,
       unsigned short dataIdentifier,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`dataIdentifier`

Defines the data identifier of the data to be read. The SAE J1979DA standard defines the values.

Output

`dataOut`

Returns the ECU data record. If you know the record data description, you can use `ndConvertFromPhys` to interpret this record. You can obtain the description from the SAE J1979DA standard.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the ndStatusToString function to obtain a descriptive string for the return value.

Description

The WWH-OBD ReadDataByIdentifier service is based on the UDS ReadDataByIdentifier service (ISO 14229-1).

ndWWHOBDRequestDTCExtendedDataRecord

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes.

Format

```
long ndWWHOBDRequestDTCExtendedDataRecord (
       TD1 *diagRef,
       unsigned char DTCMaskRecord[],
       long lenDTCMaskRecord,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCMaskRecord`

Specifies the DTC mask record.

`lenDTCMaskRecord`

Contains the number of valid data bytes in the DTCMaskRecord array.

Output

`dataOut`

Returns the ECU data record.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

ndWWHOBDRequestEmissionRelatedDTCs

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndWWHOBDRequestEmissionRelatedDTCS (
       TD1 *diagRef,
       unsigned char DTCSeverityMask,
       unsigned char DTCStatusMask,
       TD3 * DTCDescriptor,
       unsigned char *severityAvailabilityMask,
       unsigned char *statusAvailabilityMask,
       unsigned char *DTCFormatIdentifier,
       TD4 * DTCs[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCSeverityMask`

Defines the severity information of DTCs to be read. The values are application specific.

`DTCStatusMask`

Defines the status of DTCs to be read. The values are application specific.

`DTCDescriptor`

A struct that describes the DTC records the ECU delivers:

```
typedef struct {
    long DTCByteLength;
    long StatusByteLength;
    long AddDataByteLength;
    unsigned short ByteOrder;
    } TD3;
```

`DTCByteLength` indicates the number of bytes the ECU sends for each DTC. The default is 3.

`StatusByteLength` indicates the number of bytes the ECU sends for each DTC's status. The default is 1.

`AddDataByteLength` indicates the number of bytes the ECU sends for each DTC's additional data. Usually, there are no additional data, so the default is 0.

`ByteOrder` indicates the byte ordering for multibyte items:

0: MSB_FIRST (Motorola), default

1: LSB_FIRST (Intel)

This function interprets the response byte stream according to this description and returns the resulting DTC records in the `DTCs` struct array.

Output

`severityAvailabilityMask`

An application-specific value returned for all DTCs.

`statusAvailabilityMask`

An application-specific value returned for all DTCs.

`DTCFormatIdentifier`

An application-specific value returned for all DTCs.

`DTCs`

Returns the resulting DTCs as an array of structs:

```
typedef struct {
   unsigned long DTC;
   unsigned long Status;
   unsigned long AddData;
   } TD4;
```

`DTC` is the resulting Diagnostic Trouble Code. You can use `ndWWHOBDConvertDTCsToJ1939` or `ndWWHOBDConvertDTCsToJ2012` to convert this to readable format as defined by SAE J1939 and SAE J2012.

`Status` is the DTC status. Usually, this is a bit field with following meaning:

Bit	Meaning
0	testFailed
1	testFailedThisOperationCycle
2	pendingDTC
3	confirmedDTC
4	testNotCompletedSinceLastClear
5	testFailedSinceLastClear

6 testNotCompletedThisOperationCycle

7 warningIndicatorRequested

`AddData` contains optional additional data for this DTC. Usually, this does not contain valid information (refer to `DTCDescriptor`).

`len`

On input, `len` must contain the `DTCs` array length in elements. On return, it contains the number of valid elements in the `DTCs` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

ndWWHOBDRequestFreezeFrameInformation

Purpose

Executes the WWH-OBD ReadDTCInformation service. Reads selected Diagnostic Trouble Codes (DTCs).

Format

```
long ndWWHOBDRequestFreezeFrameInformation (
       TD1 *diagRef,
       unsigned char DTCMaskRecord[],
       long lenDTCMaskRecord,
       unsigned char DTCRecordNumber,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DTCMaskRecord`

Specifies the DTC mask record.

`lenDTCMaskRecord`

Contains the number of valid data bytes in the `DTCMaskRecord` array.

`DTCRecordNumber`

Specifies the snapshot record number.

Output

`dataOut`

Returns the ECU data record.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD ReadDTCInformation service is based on the UDS ReaDTCInformation service (ISO 14229-1).

ndWWHOBDRequestRID

Purpose

Executes the WWH-OBD RoutineControl service. Reads a data record from the ECU.

Format

```
long ndWWHOBDRequestRID (
       TD1 *diagRef,
       unsigned char mode,
       unsigned short routineIdentifier,
       unsigned char dataOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`mode`

Defines the service operation mode. You can obtain the values from a ring control:

1: Start Routine

2: Stop Routine

3: Request Routine Results

Other values are application specific.

`routineIdentifier`

Defines the identifier of the routine to be started. The values are application specific.

Output

`dataOut`

Returns application-specific output parameters from the routine.

`len`

On input, `len` must contain the `dataOut` array length. On return, it contains the number of valid data bytes in the `dataOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD RoutineControl service is based on the UDS RoutineControl service (ISO 14229-1).

ndWWHOBDRequestSupportedDIDs

Purpose

Executes the WWH-OBD ReadDataByIdentifier service to retrieve the valid DID values for this service.

Format

```
long ndWWHOBDRequestSupportedDIDs (
       TD1 *diagRef,
       unsigned short DID,
       unsigned char DIDsOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

`DID`

Specifies the diagnostic data identifier for this service. The following values are valid and can be obtained through an enum control:

0 **PID:** parameter identifier

1 **MID:** monitor identifier

2 **ITID:** info type identifier

Output

`DIDsOut`

Returns an array of valid DIDs.

`len`

On input, `len` must contain the `DIDsOut` array length. On return, it contains the number of valid data bytes in the `DIDsOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD ReadDataByIdentifier service is based on the UDS ReadDataByIdentifier service (ISO 14229-1).

ndWWHOBDRequestSupportedRIDs

Purpose

Executes the WWH-OBD RoutineControl service to retrieve the valid RID values for this service.

Format

```
long ndWWHOBDRequestSupportedRIDs (
       TD1 *diagRef,
       unsigned char RIDsOut[],
       long *len,
       LVBoolean *success);
```

Input

`diagRef`

Specifies the diagnostic session handle, obtained from `ndOpenDiagnostic` and passed to subsequent diagnostic functions. Normally, it is not necessary to manually manipulate the elements of this struct.

Output

`RIDsOut`

Returns an array of valid RIDs.

`len`

On input, `len` must contain the `RIDsOut` array length. On return, it contains the number of valid data bytes in the `RIDsOut` array.

`success`

Indicates successful receipt of a positive response message for this diagnostic service.

Return Value

The return value indicates the function call status as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `ndStatusToString` function to obtain a descriptive string for the return value.

Description

The WWH-OBD RoutineControl service is based on the UDS RoutineControl service (ISO 14229-1).

A

NI Services

National Instruments provides global services and support as part of our commitment to your success. Take advantage of product services in addition to training and certification programs that meet your needs during each phase of the application life cycle; from planning and development through deployment and ongoing maintenance.

To get started, register your product at ni.com/myproducts.

As a registered NI product user, you are entitled to the following benefits:

- Access to applicable product services.
- Easier product management with an online account.
- Receive critical part notifications, software updates, and service expirations.

Log in to your National Instruments ni.com User Profile to get personalized access to your services.

Services and Resources

- **Maintenance and Hardware Services—**NI helps you identify your systems' accuracy and reliability requirements and provides warranty, sparing, and calibration services to help you maintain accuracy and minimize downtime over the life of your system. Visit ni.com/services for more information**.**
  - **Warranty and Repair—**All NI hardware features a one-year standard warranty that is extendable up to five years. NI offers repair services performed in a timely manner by highly trained factory technicians using only original parts at a National Instruments service center.
  - **Calibration—**Through regular calibration, you can quantify and improve the measurement performance of an instrument. NI provides state-of-the-art calibration services. If your product supports calibration, you can obtain the calibration certificate for your product at ni.com/calibration.

- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.
- **Training and Certification—**The NI training and certification program is the most effective way to increase application development proficiency and productivity. Visit ni.com/training for more information.
  - The Skills Guide assists you in identifying the proficiency requirements of your current application and gives you options for obtaining those skills consistent with your time and budget constraints and personal learning preferences. Visit ni.com/skills-guide to see these custom paths.
  - NI offers courses in several languages and formats including instructor-led classes at facilities worldwide, courses on-site at your facility, and online courses to serve your individual needs.
- **Technical Support—**Support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources—**Visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Software Support Service Membership—**The Standard Service Program (SSP) is a renewable one-year subscription included with almost every NI software product, including NI Developer Suite. This program entitles members to direct access to NI Applications Engineers through phone and email for one-to-one technical support, as well as exclusive access to online training modules at ni.com/self-paced-training. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit ni.com/ssp for more information.
- **Declaration of Conformity (DoC)—**A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product

safety. You can obtain the DoC for your product by visiting ni.com/certification.

For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.

You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

A

activating your software, *xvii*
application development, 3-1
    on CompactRIO or R Series using NI 985x or NI 986x C Series module, 3-4
Automotive Diagnostic Command Set
    API
        C, 6-1
        LabVIEW, 5-1
    API structure, 4-2
    application development, 3-1
    available diagnostic services, 4-4
    choosing a programming language, 3-1
    configuration, 2-1
    general programming model (figure), 4-3
    hardware requirements, 2-3
    installation, 2-1
    introduction, 1-1
    KWP2000, 1-1
        connect/disconnect, 1-3
        diagnostic service format, 1-2
        diagnostic services, 1-2
        Diagnostic Trouble Codes, 1-4
        external references, 1-4
        GetSeed/Unlock, 1-3
        input/output control, 1-4
        measurements, 1-4
        read/write memory, 1-3
        remote activation of a routine, 1-4
        transport protocol, 1-2
    LabVIEW RT configuration, 2-2
    OBD, 1-6
    software requirements, 2-3
    structure (figure), 4-1
    tweaking the transport protocol, 4-4
    UDS, 1-5
        diagnostic service format, 1-6
        diagnostic services, 1-5
        external references, 1-6
    using, 4-1
    using with LabVIEW, 3-1
    using with LabWindows/CVI, 3-1
    using with other programming languages, 3-3
    using with Visual C++ 6, 3-2
available diagnostic services, 4-4

C

C API
    DoIP functions, 6-50
    general functions, 6-13
    KWP2000 services, 6-63
    list of data types, 6-2
    list of functions, 6-3
    ndClearDiagnosticInformation, 6-63
    ndCloseDiagnostic, 6-13
    ndControlDTCSetting, 6-65
    ndConvertFromPhys, 6-14
    ndConvertToPhys, 6-16
    ndCreateExtendedCANIds, 6-18
    ndDiagFrameRecv, 6-20
    ndDiagFrameSend, 6-22
    ndDiagnosticService, 6-23
    ndDisableNormalMessage Transmission, 6-67
    ndDoIPActivateRouting, 6-50
    ndDoIPConnect, 6-52
    ndDoIPDisconnect, 6-54
    ndDoIPEntityStatus, 6-55
    ndDoIPGetDiagPowerMode, 6-57
    ndDoIPGetEntities, 6-58

ndDoIPSendVehicleIdentReqEID, 6-61
ndDoIPSendVehicleIdentRequest, 6-60
ndDoIPSendVehicleIdentReqVIN, 6-62
ndDTCToString, 6-25
ndECUReset, 6-68
ndEnableNormalMessage Transmission, 6-70
ndGetProperty, 6-26
ndGetTimeStamp, 6-29
ndInputOutputControlByLocal Identifier, 6-71
ndOBDClearEmissionRelatedDiagnostic Information, 6-153
ndOBDOpen, 6-30
ndOBDRequestControlOfOnBoard Device, 6-154
ndOBDRequestCurrentPowertrain DiagnosticData, 6-156
ndOBDRequestEmissionRelated DTCs, 6-158
ndOBDRequestEmissionRelatedDTCs DuringCurrentDriveCycle, 6-160
ndOBDRequestOnBoardMonitoringTest Results, 6-162
ndOBDRequestPermanentFault Codes, 6-164
ndOBDRequestPowertrainFreezeFrame Data, 6-166
ndOBDRequestVehicle Information, 6-168
ndOpenDiagnostic, 6-33
ndOpenDiagnosticOnIP, 6-37
ndOpenDiagnosticOnLIN, 6-39
ndReadDataByLocalIdentifier, 6-73
ndReadDTCByStatus, 6-75
ndReadECUIdentification, 6-78
ndReadMemoryByAddress, 6-80
ndReadStatusOfDTC, 6-82
ndRequestRoutineResultsByLocal Identifier, 6-85
ndRequestSeed, 6-87
ndSendKey, 6-89
ndSetProperty, 6-41
ndStartDiagnosticSession, 6-91
ndStartRoutineByLocalIdentifier, 6-92
ndStatusToString, 6-44
ndStopDiagnosticSession, 6-94
ndStopRoutineByLocalIdentifier, 6-95
ndTesterPresent, 6-97
ndUDS06ReadMemoryBy Address, 6-149
ndUDS06WriteMemoryBy Address, 6-151
ndUDSClearDiagnostic Information, 6-103
ndUDSCommunicationControl, 6-105
ndUDSControlDTCSetting, 6-107
ndUDSDiagnosticSessionControl, 6-108
ndUDSECUReset, 6-109
ndUDSInputOutputControlBy Identifier, 6-111
ndUDSReadDataByIdentifier, 6-113
ndUDSReadMemoryByAddress, 6-115
ndUDSReportDTCBySeverityMask Record, 6-117
ndUDSReportDTCByStatusMask, 6-120
ndUDSReportSeverityInformationOf DTC, 6-123
ndUDSReportSupportedDTCs, 6-126
ndUDSRequestDownload, 6-129
ndUDSRequestSeed, 6-131
ndUDSRequestTransferExit, 6-133
ndUDSRequestUpload, 6-135
ndUDSRoutineControl, 6-137
ndUDSSendKey, 6-139
ndUDSTesterPresent, 6-141
ndUDSTransferData, 6-143
ndUDSWriteDataByIdentifier, 6-145
ndUDSWriteMemoryByAddress, 6-147
ndVWTPConnect, 6-46
ndVWTPConnectionTest, 6-48
ndVWTPDisconnect, 6-49

ndWriteDataByLocalIdentifier, 6-99
ndWriteMemoryByAddress, 6-101
ndWWHOBDClearEmissionRelated DTCs, 6-170
ndWWHOBDConvertDTCsTo J1939, 6-171
ndWWHOBDConvertDTCsTo J2012, 6-173
ndWWHOBDRequestDID, 6-175
ndWWHOBDRequestDTCExtendedData Record, 6-177
ndWWHOBDRequestEmissionRelated DTCs, 6-179
ndWWHOBDRequestFreezeFrame Information, 6-182
ndWWHOBDRequestRID, 6-184
ndWWHOBDRequestSupported DIDs, 6-186
ndWWHOBDRequestSupported RIDs, 6-188
OBD (On-Board Diagnostics) services, 6-153
UDS (DiagOnCAN) services, 6-103
WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) services, 6-170
ClearDiagnosticInformation.vi, 5-70
Close Diagnostic.vi, 5-10
CompactRIO
application development on using NI 985x or NI 986x C Series module, 3-4
computer ID, *xviii*
configuration, 2-1
connect/disconnect, KWP2000, 1-3
ControlDTCSetting.vi, 5-73
Convert from Phys.vi, 5-12
Convert to Phys.vi, 5-14
Create Extended CAN IDs.vi, 5-16

D

deactivating a product, *xix*
Diag Get Property.vi, 5-17
Diag Set Property.vi, 5-20
Diagnostic Frame Recv.vi, 5-23
Diagnostic Frame Send.vi, 5-25
diagnostic service format
KWP2000, 1-2
UDS, 1-6
Diagnostic Service.vi, 5-27
diagnostic services
available, 4-4
KWP2000, 1-2
UDS, 1-5
Diagnostic Trouble Codes
KWP2000, 1-4
DisableNormalMessageTransmission.vi, 5-76
documentation
NI resources, A-1
related documentation, *xv*
DoIP Activate Routing.vi, 5-51
DoIP Connect.vi, 5-53
DoIP Disconnect.vi, 5-55
DoIP functions
C API, 6-50
LabVIEW API, 5-51
DoIP Get Diagnostic Power Mode.vi, 5-57
DoIP Get DoIP Entity Status.vi, 5-59
DoIP Get Entities.vi, 5-61
DoIP Send Vehicle Identification Request w EID.vi, 5-66
DoIP Send Vehicle Identification Request w VIN.vi, 5-68
DoIP Send Vehicle Identification Request.vi, 5-64
DTC to String.vi, 5-29

E

ECUReset.vi, 5-78
EnableNormalMessageTransmission.vi, 5-80
external references
    KWP2000, 1-4
    UDS, 1-6

G

general functions
    C API, 6-13
    LabVIEW API, 5-10
general programming model (figure), 4-3
Get Time Stamp.vi, 5-30
GetSeed/Unlock, 1-3

H

hardware requirements, 2-3
help, technical support, A-1

I

input/output control, 1-4
InputOutputControlByLocalIdentifier.vi, 5-82
installation, 2-1
introduction, 1-1

K

Key Word Protocol 2000, 1-1
KWP2000
    connect/disconnect, 1-3
    definition, 1-1
    diagnostic service format, 1-2
    diagnostic services, 1-2
    Diagnostic Trouble Codes, 1-4
    external references, 1-4
    GetSeed/Unlock, 1-3
    input/output control, 1-4
    measurements, 1-4
    read/write memory, 1-3
    remote activation of a routine, 1-4
    transport protocol, 1-2
KWP2000 services
    C API, 6-63
    LabVIEW API, 5-70

L

LabVIEW
    using with Automotive Diagnostic Command Set, 3-1
LabVIEW API
    ClearDiagnosticInformation.vi, 5-70
    Close Diagnostic.vi, 5-10
    ControlDTCSetting.vi, 5-73
    Convert from Phys.vi, 5-12
    Convert to Phys.vi, 5-14
    Create Extended CAN IDs.vi, 5-16
    Diag Get Property.vi, 5-17
    Diag Set Property.vi, 5-20
    Diagnostic Frame Recv.vi, 5-23
    Diagnostic Frame Send.vi, 5-25
    Diagnostic Service.vi, 5-27
    DisableNormalMessage Transmission.vi, 5-76
    DoIP Activate Routing.vi, 5-51
    DoIP Connect.vi, 5-53
    DoIP Disconnect.vi, 5-55
    DoIP functions, 5-51
    DoIP Get Diagnostic Power Mode.vi, 5-57
    DoIP Get DoIP Entity Status.vi, 5-59
    DoIP Get Entities.vi, 5-61
    DoIP Send Vehicle Identification Request w EID.vi, 5-66
    DoIP Send Vehicle Identification Request w VIN.vi, 5-68
    DoIP Send Vehicle Identification Request.vi, 5-64
    DTC to String.vi, 5-29

ECUReset.vi, 5-78
EnableNormalMessage Transmission.vi, 5-80
general functions, 5-10
Get Time Stamp.vi, 5-30
InputOutputControlByLocal Identifier.vi, 5-82
KWP2000 services, 5-70
list of VIs, 5-2
OBD (On-Board Diagnostics) services, 5-170
OBD Clear Emission Related Diagnostic Information.vi, 5-170
OBD Open.vi, 5-32
OBD Request Control Of On-Board Device.vi, 5-172
OBD Request Current Powertrain Diagnostic Data.vi, 5-174
OBD Request Emission Related DTCs During Current Drive Cycle.vi, 5-179
OBD Request Emission Related DTCs.vi, 5-176
OBD Request On-Board Monitoring Test Results.vi, 5-182
OBD Request Permanent Fault Codes.vi, 5-184
OBD Request Powertrain Freeze Frame Data.vi, 5-187
OBD Request Supported PIDs.vi, 5-189
OBD Request Vehicle Information.vi, 5-191
Open Diagnostic on IP.vi, 5-40
Open Diagnostic on LIN.vi, 5-42
Open Diagnostic.vi, 5-36
ReadDataByLocalIdentifier.vi, 5-84
ReadDTCByStatus.vi, 5-86
ReadECUIdentification.vi, 5-89
ReadMemoryByAddress.vi, 5-91
ReadStatusOfDTC.vi, 5-93
RequestRoutineResultsByLocal Identifier.vi, 5-96
RequestSeed.vi, 5-98
SendKey.vi, 5-100
StartDiagnosticSession.vi, 5-102
StartRoutineByLocalIdentifier.vi, 5-104
StopDiagnosticSession.vi, 5-106
StopRoutineByLocalIdentifier.vi, 5-108
TesterPresent.vi, 5-110
UDS (DiagOnCAN) services, 5-116
UDS ClearDiagnostic Information.vi, 5-116
UDS CommunicationControl.vi, 5-119
UDS ControlDTCSetting.vi, 5-121
UDS DiagnosticSessionControl.vi, 5-123
UDS ECUReset.vi, 5-125
UDS InputOutputControlBy Identifier.vi, 5-127
UDS ReadDataByIdentifier.vi, 5-129
UDS ReadMemoryByAddress.vi, 5-131
UDS ReportDTCBySeverityMask Record.vi, 5-133
UDS ReportDTCByStatusMask.vi, 5-136
UDS ReportSeverityInformationOf DTC.vi, 5-139
UDS ReportSupportedDTCs.vi, 5-142
UDS RequestDownload.vi, 5-145
UDS RequestSeed.vi, 5-147
UDS RequestTransferExit.vi, 5-149
UDS RequestUpload.vi, 5-151
UDS RoutineControl.vi, 5-153
UDS SendKey.vi, 5-155
UDS TesterPresent.vi, 5-157
UDS TransferData.vi, 5-159
UDS WriteDataByIdentifier.vi, 5-162
UDS WriteMemoryByAddress.vi, 5-164
UDS06 ReadMemoryBy Address.vi, 5-166
UDS06 WriteMemoryBy Address.vi, 5-168
VWTP Connect.vi, 5-45
VWTP Connection Test.vi, 5-47
VWTP Disconnect.vi, 5-49

WriteDataByLocalIdentifier.vi, 5-112
WriteMemoryByAddress.vi, 5-114
WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) services, 5-193
WWH-OBD Clear Emission Related DTCs.vi, 5-193
WWH-OBD Convert DTCs to J1939.vi, 5-195
WWH-OBD Convert DTCs to J2012.vi, 5-197
WWH-OBD Request DID.vi, 5-199
WWH-OBD Request DTC Extended Data Record.vi, 5-201
WWH-OBD Request Emission Related DTCs.vi, 5-203
WWH-OBD Request Freeze Frame Information.vi, 5-206
WWH-OBD Request RID.vi, 5-208
WWH-OBD Request Supported DIDs.vi, 5-210
WWH-OBD Request Supported RIDs.vi, 5-212
LabVIEW RT configuration, 2-2
LabWindows/CVI
using with Automotive Diagnostic Command Set, 3-1
list of C functions, 6-3
list of data types, 6-2
list of LabVIEW VIs, 5-2

N

ndClearDiagnosticInformation, 6-63
ndCloseDiagnostic, 6-13
ndControlDTCSetting, 6-65
ndConvertFromPhys, 6-14
ndConvertToPhys, 6-16
ndCreateExtendedCANIds, 6-18
ndDiagFrameRecv, 6-20
ndDiagFrameSend, 6-22
ndDiagnosticService, 6-23
ndDisableNormalMessageTransmission, 6-67
ndDoIPActivateRouting, 6-50
ndDoIPConnect, 6-52
ndDoIPDisconnect, 6-54
ndDoIPEntityStatus, 6-55
ndDoIPGetDiagPowerMode, 6-57
ndDoIPGetEntities, 6-58
ndDoIPSendVehicleIdentReqEID, 6-61
ndDoIPSendVehicleIdentRequest, 6-60
ndDoIPSendVehicleIdentReqVIN, 6-62
ndDTCToString, 6-25
ndECUReset, 6-68
ndEnableNormalMessageTransmission, 6-70
ndGetProperty, 6-26
ndGetTimeStamp, 6-29
ndInputOutputControlByLocalIdentifier, 6-71
ndOBDClearEmissionRelatedDiagnostic Information, 6-153
ndOBDOpen, 6-30
ndOBDRequestControlOfOnBoard Device, 6-154
ndOBDRequestCurrentPowertrainDiagnostic Data, 6-156
ndOBDRequestEmissionRelatedDTCs, 6-158
ndOBDRequestEmissionRelatedDTCsDuring CurrentDriveCycle, 6-160
ndOBDRequestOnBoardMonitoringTest Results, 6-162
ndOBDRequestPermanentFaultCodes, 6-164
ndOBDRequestPowertrainFreezeFrame Data, 6-166
ndOBDRequestVehicleInformation, 6-168
ndOpenDiagnostic, 6-33
ndOpenDiagnosticOnIP, 6-37
ndOpenDiagnosticOnLIN, 6-39
ndReadDataByLocalIdentifier, 6-73
ndReadDTCByStatus, 6-75
ndReadECUIdentification, 6-78
ndReadMemoryByAddress, 6-80
ndReadStatusOfDTC, 6-82

ndRequestRoutineResultsByLocal Identifier, 6-85
ndRequestSeed, 6-87
ndSendKey, 6-89
ndSetProperty, 6-41
ndStartDiagnosticSession, 6-91
ndStartRoutineByLocalIdentifier, 6-92
ndStatusToString, 6-44
ndStopDiagnosticSession, 6-94
ndStopRoutineByLocalIdentifier, 6-95
ndTesterPresent, 6-97
ndUDS06ReadMemoryByAddress, 6-149
ndUDS06WriteMemoryByAddress, 6-151
ndUDSClearDiagnosticInformation, 6-103
ndUDSCommunicationControl, 6-105
ndUDSControlDTCSetting, 6-107
ndUDSDiagnosticSessionControl, 6-108
ndUDSECUReset, 6-109
ndUDSInputOutputControlBy Identifier, 6-111
ndUDSReadDataByIdentifier, 6-113
ndUDSReadMemoryByAddress, 6-115
ndUDSReportDTCBySeverityMask Record, 6-117
ndUDSReportDTCByStatusMask, 6-120
ndUDSReportSeverityInformationOf DTC, 6-123
ndUDSReportSupportedDTCs, 6-126
ndUDSRequestDownload, 6-129
ndUDSRequestSeed, 6-131
ndUDSRequestTransferExit, 6-133
ndUDSRequestUpload, 6-135
ndUDSRoutineControl, 6-137
ndUDSSendKey, 6-139
ndUDSTesterPresent, 6-141
ndUDSTransferData, 6-143
ndUDSWriteDataByIdentifier, 6-145
ndUDSWriteMemoryByAddress, 6-147
ndVWTPConnect, 6-46
ndVWTPConnectionTest, 6-48
ndVWTPDisconnect, 6-49
ndWriteDataByLocalIdentifier, 6-99
ndWriteMemoryByAddress, 6-101
ndWWHOBDClearEmissionRelated DTCs, 6-170
ndWWHOBDConvertDTCsToJ1939, 6-171
ndWWHOBDConvertDTCsToJ2012, 6-173
ndWWHOBDRequestDID, 6-175
ndWWHOBDRequestDTCExtendedData Record, 6-177
ndWWHOBDRequestEmissionRelated DTCs, 6-179
ndWWHOBDRequestFreezeFrame Information, 6-182
ndWWHOBDRequestRID, 6-184
ndWWHOBDRequestSupportedDIDs, 6-186
ndWWHOBDRequestSupportedRIDs, 6-188
NI Activation Wizard, *xvii*

O

OBD, 1-6
OBD (On-Board Diagnostics) services
C API, 6-153
LabVIEW API, 5-170
OBD Clear Emission Related Diagnostic Information.vi, 5-170
OBD Open.vi, 5-32
OBD Request Control Of On-Board Device.vi, 5-172
OBD Request Current Powertrain Diagnostic Data.vi, 5-174
OBD Request Emission Related DTCs During Current Drive Cycle.vi, 5-179
OBD Request Emission Related DTCs.vi, 5-176
OBD Request On-Board Monitoring Test Results.vi, 5-182
OBD Request Permanent Fault Codes.vi, 5-184
OBD Request Powertrain Freeze Frame Data.vi, 5-187
OBD Request Supported PIDs.vi, 5-189

OBD Request Vehicle Information.vi, 5-191
On-Board Diagnostic, 1-6
Open Diagnostic on IP.vi, 5-40
Open Diagnostic on LIN.vi, 5-42
Open Diagnostic.vi, 5-36
other programming languages, using with
Automotive Diagnostic Command Set, 3-3

P

programming language, choosing, 3-1

R

R Series
application development on using
NI 985x or NI 986x C Series
module, 3-4
read/write memory, 1-3
ReadDataByLocalIdentifier.vi, 5-84
ReadDTCByStatus.vi, 5-86
ReadECUIdentification.vi, 5-89
ReadMemoryByAddress.vi, 5-91
ReadStatusOfDTC.vi, 5-93
related documentation, *xv*
remote action of a routine, KWP2000, 1-4
RequestRoutineResultsByLocal
Identifier.vi, 5-96
RequestSeed.vi, 5-98

S

SendKey.vi, 5-100
serial number, finding, *xviii*
software
activating, *xvii*
evaluating, *xix*
moving after activation, *xix*
requirements, 2-3
StartDiagnosticSession.vi, 5-102
StartRoutineByLocalIdentifier.vi, 5-104
StopDiagnosticSession.vi, 5-106
StopRoutineByLocalIdentifier.vi, 5-108
support, technical, A-1

T

technical support, A-1
TesterPresent.vi, 5-110
transport protocol
KWP2000, 1-2
tweaking, 4-4
tweaking the transport protocol, 4-4

U

UDS, 1-5
diagnostic service format, 1-6
diagnostic services, 1-5
external references, 1-6
UDS (DiagOnCAN) services
C API, 6-103
LabVIEW API, 5-116
UDS ClearDiagnosticInformation.vi, 5-116
UDS CommunicationControl.vi, 5-119
UDS ControlDTCSetting.vi, 5-121
UDS DiagnosticSessionControl.vi, 5-123
UDS ECUReset.vi, 5-125
UDS InputOutputControlBy
Identifier.vi, 5-127
UDS ReadDataByIdentifier.vi, 5-129
UDS ReadMemoryByAddress.vi, 5-131
UDS ReportDTCBySeverityMask
Record.vi, 5-133
UDS ReportDTCByStatusMask.vi, 5-136
UDS ReportSeverityInformationOf
DTC.vi, 5-139
UDS ReportSupportedDTCs.vi, 5-142
UDS RequestDownload.vi, 5-145
UDS RequestSeed.vi, 5-147
UDS RequestTransferExit.vi, 5-149
UDS RequestUpload.vi, 5-151

UDS RoutineControl.vi, 5-153
UDS SendKey.vi, 5-155
UDS TesterPresent.vi, 5-157
UDS TransferData.vi, 5-159
UDS WriteDataByIdentifier.vi, 5-162
UDS WriteMemoryByAddress.vi, 5-164
UDS06 ReadMemoryByAddress.vi, 5-166
UDS06 WriteMemoryByAddress.vi, 5-168
Unified Diagnostic Services, 1-5

V

Visual C++ 6, using with Automotive Diagnostic Command Set, 3-2
VWTP Connect.vi, 5-45
VWTP Connection Test.vi, 5-47
VWTP Disconnect.vi, 5-49

W

Web resources, A-1
Windows Guest accounts, *xix*
WriteDataByLocalIdentifier.vi, 5-112
WriteMemoryByAddress.vi, 5-114
WWH-OBD (World-Wide-Harmonized On-Board Diagnostics) services
  C API, 6-170
  LabVIEW API, 5-193
WWH-OBD Clear Emission Related DTCs.vi, 5-193
WWH-OBD Convert DTCs to J1939.vi, 5-195
WWH-OBD Convert DTCs to J2012.vi, 5-197
WWH-OBD Request DID.vi, 5-199
WWH-OBD Request DTC Extended Data Record.vi, 5-201
WWH-OBD Request Emission Related DTCs.vi, 5-203
WWH-OBD Request Freeze Frame Information.vi, 5-206
WWH-OBD Request RID.vi, 5-208
WWH-OBD Request Supported DIDs.vi, 5-210
WWH-OBD Request Supported RIDs.vii, 5-212